DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Databases Project – Spring 2019

Team No: 53
Names: Sadra Boreiri, Saleh Gholamzadeh, Timur Lavrov

Contents

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

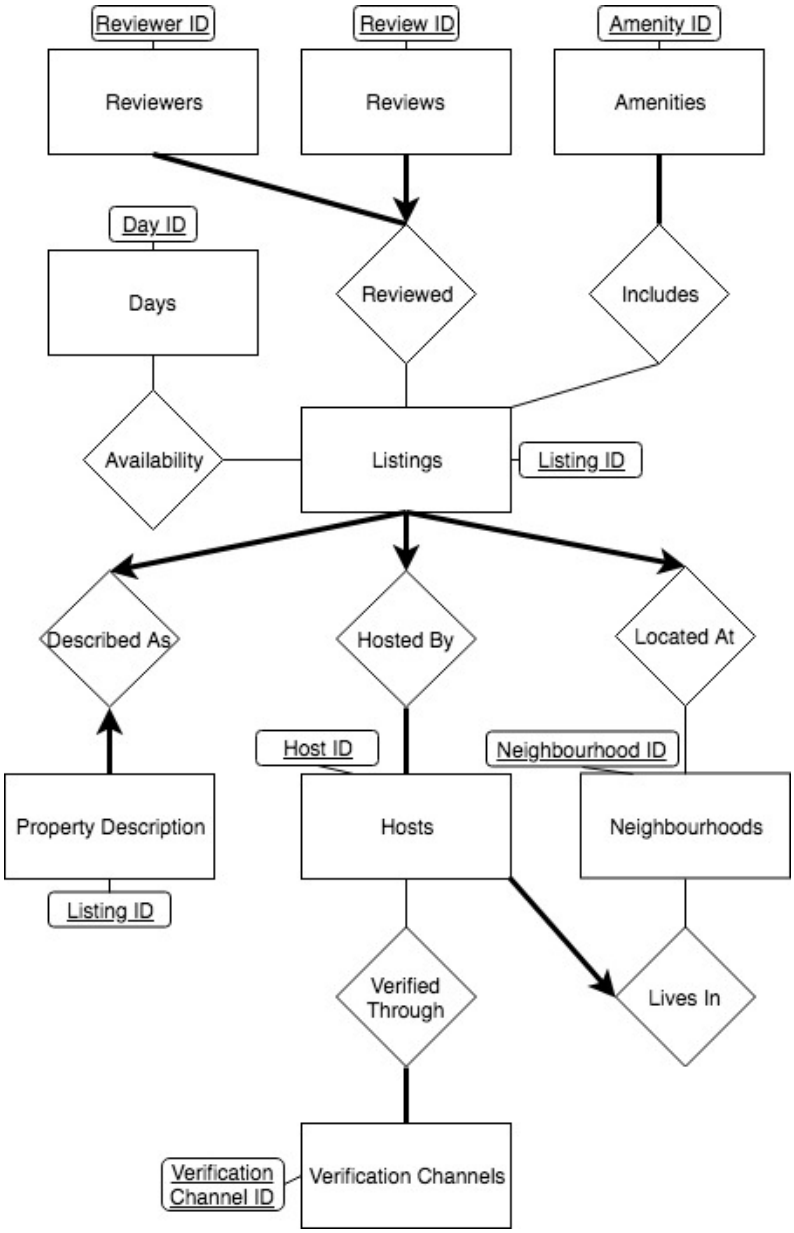
Deliverable 1

Assumptions

We assumed that the data provided was already in an appropriate format for the Air BnB platform. Hence, we defined our constraints in the ER Model (e.g. total participation/one-to-many) and the Relational Schema (e.g. whether a certain attribute is nullable or unique) based on what we observed in the data.

Entity Relationship Schema

Schema



DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Description

Looking at the data, we understood that the listings table would need to be fragmented into multiple entities. First of all, the central entity to our model would be the **Listings** entity representing the properties and their attributes. Secondly, we observed that the **Hosts** could have an entity of their own. Every listing is hosted by precisely one host and a given host must host at least one property but can host many. Therefore, we defined the **Hosted By** relation with the described constraints.
Subsequently, we also noticed that the representation of the data within the 'host_verification' and 'amenities' columns in the listings tables was not appropriate for database storage. Tuples in both of these columns were represented as lists and nested lists would not be practical when running SQL queries. Hence, we decided to create **Verification Channels** and **Amenities** entities. Each of these would hold the list of all of the various means to verify a host and all the amenities that exist respectively. We determined that this would be more efficient than adding a large number of attributes to the **Listings** entity which would create sparse binary data and would not make it practical to add a new verification channel or amenity. The **Verification Channels** entity will only hold channels that have been used to verify hosts. However, some hosts might not have been verified yet. Hence, we created the **Verified Through** relation between which reflects these constraints. Similarly, the **Amenities** entity only lists amenities that appear in at least one listing. However, a listing might not have any amenities. Thus, we created the **Includes** relation that illustrates this.
For the geographical data that relates to the location of the listings and the place of residency of the hosts, we created the **Neighbourhoods** entity. This entity consists of all the neighbourhoods where at least one host lives or at least one listing is located. As such, there can be neighbourhoods where there are no listings or where no hosts live, but not both. Combining this with the fact multiple hosts and listings can be located in a given neighbourhood justifies the absence of constraints for the **Lives In** and **Located At** relations.
Additionally, in the listings data we observed that there were numerous columns that stored large text data that described the listings. Since it would be costly to load this significantly large data at every query and this data would not be queried often, we decided to create a **Property Description** entity having a one-to-one relation with the **Listings** entity with total participation.
For the data contained in the calendar and reviews datasets, we represent it with the **Reviewers, Reviews/Reviewed** and **Days/Availability** entity/relations. The **Days** entity is a listing of all days over the relevant time period. We set the **Availability** relation between it and the **Listings** entity which effectively lists the combination of days and listings representing availability. Since availability is binary data we plan to drop the combination of days/listings listed as unavailable to save space. All listings can be unavailable on a given day and a listing can be available on multiple days. Hence, we set no constraints on the relation. Similarly, the **Reviewers** entity is a list of every Air BnB user that has left a review and the **Reviews** entity is a table containing all of the reviews that have been written. We set a ternary relation **Reviewed** between the **Listings**, **Reviewers** and **Reviews** entities which illustrates all of the reviews that have been submitted for all listings. Every reviewer in the **Reviewers** entity must have at least submitted one review for a listing but might have written many for one or multiple listings, which results in a total participation constraint. A review in the **Reviews** entity has to and can only be associated with one reviewer and one listing, which we illustrated with a key and total participation constraint. A listing in **Listings** might have multiple or no reviews which explains the absence of constraints.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Relational Schema

ER schema to Relational schema

In our ER model we have 9 entities and 8 relations. This translated to 15 tables in the relational schema. This is due to the fact that the **Described As** and **Hosted By** relations could be captured in the **Property Description** and **Listings** tables respectively as a result of the set constraints. All tables representing entities have their primary keys set to those illustrated in the ER model. Those relations that have their own tables have their primary keys set as the combination of the primary keys of the entities they link (as by definition). Additionally, a number of additional constraints were set based on the observed raw data and its logical interpretation. For example, we observed that every listing URL is unique in the **Listings** entity (as expected). Hence, we set a unique constraint on this column in its respective table. All of these constraints are illustrated in the DDL code provided below.

DDL

```
CREATE TABLE LISTINGS (
	listing_id INTEGER,
	listing_url VARCHAR(50) NOT NULL,
	listing_name VARCHAR(255),
	picture_url VARCHAR(255),
	host_id INTEGER NOT NULL,
	latitude DECIMAL(18,16) NOT NULL,
	longitude DECIMAL(18,16) NOT NULL,
	listing_type VARCHAR(50) NOT NULL,
	room_type VARCHAR(50) NOT NULL,
	accomodates INTEGER NOT NULL,
	bathrooms DECIMAL(3,1),
	bedrooms DECIMAL(3,1),
	beds INTEGER,
	bed_type VARCHAR(30) NOT NULL,
	square_feet INTEGER,
	price DECIMAL(10,2) NOT NULL,
	weekly_price DECIMAL(10,2),
	montly_price DECIMAL(10,2),
	security_deposit DECIMAL(10,2),
	cleaning_fee DECIMAL(8,2),
	guests_included INTEGER NOT NULL,
	extra_people DECIMAL(7,2) NOT NULL,
	minimum_nights INTEGER NOT NULL,
	maximum_nights INTEGER NOT NULL,
	review_scores_rating DECIMAL(4,1),
	review_scores_accuracy DECIMAL(3,1),
	review_scores_cleanliness DECIMAL(3,1),
```

```
	review_scores_checkin DECIMAL(3,1),
	review_scores_communication DECIMAL(3,1),
	review_scores_location DECIMAL(3,1),
	review_scores_value DECIMAL(3,1),
	is_business_travel_ready CHAR(1) NOT NULL,
	cancellation_policy VARCHAR(50) NOT NULL,
	require_guest_profile_picture CHAR(1) NOT NULL,
	require_guest_profile_picture CHAR(1) NOT NULL,
	PRIMARY KEY (listing_id),
	UNIQUE (listing_url),
	UNIQUE (longitude, latitude),
	FOREIGN KEY (host_id) REFERENCES HOSTS
)
```

```
CREATE TABLE HOSTS (
	host_id INTEGER,
	host_url VARCHAR(50) NOT NULL,
	host_name VARCHAR(255) NOT NULL,
	host_since DATE NOT NULL,
	host_about CLOB,
	host_response_time VARCHAR(50),
	host_response_rate decimal(3,2),
	CHECK (host_response_rate >=0 AND host_response_rate<=1),
	host_thumbnail_url VARCHAR(255),
	host_picture_url VARCHAR(255),
	PRIMARY KEY (host_id)
)
```

```
CREATE TABLE PROPERTY_DESCRIPTION (
	listing_id INTEGER NOT NULL,
	summary_ CLOB,
	space_ CLOB,
	description CLOB,
	neighbourhood_overview CLOB,
	notes CLOB,
	transit CLOB,
	access_ CLOB,
	interaction CLOB,
	house_rules CLOB,
	PRIMARY KEY (listing_id),
```

```
	FOREIGN KEY (listing_id) REFERENCES LISTINGS (listing_id)
)
```

```
CREATE TABLE VERIFICATION_CHANELS (
	verification_channel_id INTEGER,
	verification_channel_name VARCHAR(255) NOT NULL,
	PRIMARY KEY (verification_channel_id)
)
```

```
CREATE TABLE DAYS (
	day_id INTEGER,
	date_ DATE NOT NULL,
	PRIMARY KEY(day_id),
	UNIQUE (date_)
)
```

```
CREATE TABLE AMENITIES (
	amenity_id INTEGER,
	amenity_name VARCHAR(100) NOT NULL,
	PRIMARY KEY (amenity_id),
	UNIQUE (amenity_name)
)
```

```
CREATE TABLE REVIEWS (
	review_id INTEGER,
	review_date DATE NOT NULL,
	review CLOB NOT NULL,
	reviewer_name VARCHAR(100),
	PRIMARY KEY (review_id)
)
```

```
CREATE TABLE REVIEWERS (
	reviewer_id INTEGER
	PRIMARY KEY (reviewer_id)
)
```

```
CREATE TABLE NEIGHBOURHOODS (
	neighbourhood_id INTEGER,
	neighbourhood_name VARCHAR(50) NOT NULL,
	city VARCHAR(50) NOT NULL,
	country_code VARCHAR(3) NOT NULL,
	country VARCHAR(75) NOT NULL,
```

```
      PRIMARY KEY(neighbourhood_id)
)
```

```
CREATE TABLE VERIFIED_THROUGH (
    host_id INTEGER,
    verification_channel_id INTEGER,
    PRIMARY KEY (host_id, verification_channel_id),
    FOREIGN KEY (host_id) REFERENCES HOSTS,
    FOREIGN KEY (verification_channel_id) REFERENCES VERIFICATION_CHANNELS
)
```

```
CREATE TABLE AVAILABILITY (
    listing_id INTEGER,
    day_id INTEGER,
    date_ DATE NOT NULL,
    price INTEGER NOT NULL,
    FOREIGN KEY (listing_id) REFERENCES LISTINGS (listing_id),
    FOREIGN KEY (day_id) REFERENCES DAYS (day_id),
    FOREIGN KEY (date_) REFERENCES DAYS (date_),
    PRIMARY KEY (listing_id, day_id)
)
```

```
CREATE TABLE INCLUDES (
    amenity_id INTEGER,
    listing_id INTEGER,
    PRIMARY KEY (amenity_id, property_id),
    FOREIGN KEY (amenity_id) REFERENCES AMENITIES (amenity_id),
    FOREIGN KEY (listing_id) REFERENCES LISTINGS (listing_id)
)
```

```
CREATE TABLE REVIEWED (
    reviewer_id INTEGER,
    listing_id INTEGER,
    review_id INTEGER,
    PRIMARY KEY (reviewer_id, listing_id, review_id),
    FOREIGN KEY (reviewer_id) REFERENCES REVIEWERS (reviewer_id),
    FOREIGN KEY (listing_id) REFERENCES LISTINGS (listing_id),
    FOREIGN KEY (review_id) REFERENCES REVIEWS (review_id)
)
```

```
CREATE TABLE LIVES_IN (
    host_id INTEGER,
```

```
    neighbourhood_id INTEGER,
    PRIMARY KEY (host_id),
    FOREIGN KEY (neighbourhood_id) REFERENCES NEIGHBOURHOODS (neighbourhood_id),
    FOREIGN KEY (host_id) REFERENCES HOSTS (host_id)
)
```

```
CREATE TABLE LOCATED_AT (
    listing_id INTEGER,
    neighbourhood_id INTEGER,
    PRIMARY KEY (listing_id),
    FOREIGN KEY (neighbourhood_id) REFERENCES NEIGHBOURHOODS (neighbourhood_id),
    FOREIGN KEY (listing_id) REFERENCES LISTINGS (listing_id)
)
```

General Comments

We included the reviewer's name in the **Reviews** table rather than the **Reviewer** one in order to allow for a reviewer to change or not display his name for different reviews.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Deliverable 2

Assumptions

We assumed that:

- All the listings provided were located in either Madrid, Barcelona or Berlin
- The data did not need to be cleaned up semantically (i.e. the data was correct)
- The identifiers that came with the data needed to be kept and shouldn't be re-created (i.e. overwriting the existing Listing ID with 0, 1, 2…., #number of listings)
- Reviewers can change names

Data Loading

Prior to loading the data into our database, we cleaned and split the dataset using Python. We split the datasets into multiple csv files. Each csv file corresponds to a table in the database. Some of the work performed as part of the clean up includes:

- Removing $ and % symbols in order to store numeric data as integers and not strings.
- Removing problematic data. For example, certain hosts appeared in more than one of the initially provided csv files. However, the information associated to a host was more complete in one csv than another. Hence, in that case we only considering the more complete tuple relating to that host.
- Removing characters unsupported in Oracle SQL from the data. For example, removing new line characters '\n' and replacing them with a whitespace.
- Parsing the data and exploding tuples containing lists into multiple rows (e.g. for the Amenities entity)
- Replacing city column for every listing to contain either Madrid, Barcelona or Berlin based on what csv it was loaded from
- Creating indices and removing duplicates where appropriate
- Removing non-availability information from the availability data in order to only lists those days on which a listing is available.

Query Implementation

Query 1:
What is the average price for a listing with 8 bedrooms?

Logic description:
Average the price of all listings that have 8 beds and round the result to 2 decimal places.

SQL:

```
SELECT ROUND(AVG(PRICE),2) as average_price FROM LISTINGS WHERE BEDS = 8
```

	AVERAGE_PRICE
1	228.4

Query 2:
What is the average cleaning review score for listings with TV?

Logic description:
Equi-join the Amenities, Includes and Listings tables. Filter to those that have 'TV' as an amenity. Average the review_scores_cleanliness for all resulting listings and round to 2 decimal places.

SQL:

```
SELECT ROUND(AVG(l.review_scores_cleanliness), 2) as average_cleanliness_score
FROM listings l, amenities a, includes i
WHERE a.amenity_id = i.amenity_id
AND i.listing_id = l.listing_id
AND a.amenity_name = 'TV'
```

AVERAGE_CLEANLINESS_SCORE
9.4

Query 3:
Print all the hosts who have an available property between date 03.2019 and 09.2019.

Logic description:
Equi-join the Listings and Availability tables. Select the distinct host ids where a listing has an availability within the given time period.

SQL:

```
SELECT DISTINCT l.host_id
FROM listings l, availability a
WHERE l.listing_id = a.listing_id
AND a.date_ >= TO_DATE('2019-03-01', 'YYYY-MM-DD')
AND a.date_ < TO_DATE('2019-10-01', 'YYYY-MM-DD')
```

	HOST_ID
1	108310
2	195624524
3	135703
4	152232
5	323105

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query 4:
Print how many listing items exist that are posted by two different hosts but the hosts have the same name.

Logic description:
Count the number of listings that are hosted by hosts that share their name with another host. The list of such hosts is computed by performing a cartesian product between the Hosts table with itself and filtering to the combinations of Hosts where the host_id is different and the names match.

SQL:

```
SELECT COUNT (*) as number_of_listings
FROM LISTINGS l
WHERE l.host_id IN (
    SELECT DISTINCT h1.host_id
    FROM HOSTS h1, HOSTS h2
    WHERE h1.host_id <> h2.host_id
    AND h1.host_name = h2.host_name)
```

NUMBER_OF_LISTINGS
30393

Query 5:
Print all the dates that 'Viajes Eco' has available accommodations for rent.

Logic description:
Select distinct dates from the availability table where the listing id is contained within the list of listings that are hosted by ‘Viajes Eco’. This listings hosted by ‘Viajes Eco’ is obtained by equi-joining the Listings and Hosts tables on host_id and filtering host_name to ‘Viajes Eco’.

SQL:

```
SELECT DISTINCT a.DATE_ as viajes_eco_available_dates
FROM availability a
WHERE a.listing_id IN (
    SELECT l.listing_id
    FROM listings l, hosts h
    WHERE h.host_name = 'Viajes Eco'
    AND l.host_id = h.host_id)
```

VIAJES_ECO_AVAILABLE_DATES
09-NOV-18
17-NOV-18
24-DEC-18
22-JAN-19
02-FEB-19

Query 6:
Find all the hosts (host_ids, host_names) that have only one listing.

Logic description:
Equi-join the Listings and Hosts table on host_id and select the host_id, host_name combinations that appear only once.

SQL:

```
SELECT l.host_id, h.host_name
FROM listings l, hosts h
WHERE l.host_id = h.host_id
GROUP BY l.host_id, h.host_name
HAVING COUNT (*) = 1
```

HOST_ID	HOST_NAME
266650	Vanessa
488151	Pols
474355	Annais
2181314	Javier
2204578	Francesca

Query 7:
What is the difference in the average price of listings with and without Wifi.

Logic description:
Obtain the average price of listings with and without wifi applying the same logic as in Query 2. From each of resulting relations, select the average price and compute the difference.

SQL:

```
SELECT a.average_price_wifi - b.average_price_no_wifi as price_diff
FROM
    (
    SELECT ROUND(AVG(l1.price), 2) as average_price_wifi
    FROM listings l1, amenities a, includes i
    WHERE a.amenity_id = i.amenity_id
    AND a.amenity_name = 'Wifi'
    AND i.listing_id = l1.listing_id
    ) a,
    (SELECT ROUND(AVG(l2.price), 2) as average_price_no_wifi
    FROM listings l2
    WHERE l2.listing_id NOT IN (
        SELECT i2.listing_id
        FROM amenities a2, includes i2
```

```
        WHERE a2.amenity_id = i2.amenity_id
        AND a2.amenity_name = 'Wifi')
    ) b
```

AVERAGE_PRICE_WIFI	AVERAGE_PRICE_NO_WIFI	DIFF
86.3	83.09	3.21

Query 8:

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

Logic description:

For both Berlin and Madrid, equi-join the Listings, Neighbourhoods and Located_At tables. Filter to listings with 8 beds and to the appropriate location (Berlin or Madrid) and average the price. From each of the resulting relations, select the average price and compute the difference.

SQL:

```
SELECT be.average_price_berlin - ma.average_price_madrid as price_diff
FROM
    (
    SELECT ROUND(AVG(l1.PRICE),2) as average_price_berlin
    FROM listings l1, neighbourhoods n1, located_at la1
    WHERE l1.listing_id = la1.listing_id
    AND la1.neighbourhood_id = n1.neighbourhood_id
    AND n1.city = 'Berlin'
    AND l1.BEDS = 8
    ) be,
    (
    SELECT ROUND(AVG(l2.PRICE),2) as average_price_madrid
    FROM listings l2, neighbourhoods n2, located_at la2
    WHERE l2.listing_id = la2.listing_id
    AND la2.neighbourhood_id = n2.neighbourhood_id
    AND n2.city = 'Madrid'
    AND l2.BEDS = 8
    ) ma
```

AVERAGE_PRICE_BERLIN	AVERAGE_PRICE_MADRID	DIFF
132.68	234.27	-101.59

Query 9:

Find the top-10 (in terms of the number of listings) hosts (host_ids, host_names) in Spain.

Logic description:
Equi-join the Listings, Hosts, Neighbourhoods and Located_At tables. Filter to listings in Spain. Group listings by hosts performing a count of each group. Assign rank by decreasing order of count and select top 10 ranking hosts.

SQL:

```
SELECT m.host_id, m.host_name, m.Rank
FROM
    (
    SELECT l.host_id, h.host_name, COUNT(*), RANK() OVER (ORDER BY COUNT(*) DESC) Rank
    FROM listings l, hosts h, located_at la, neighbourhoods n
    WHERE l.host_id = h.host_id
    AND l.listing_id = la.listing_id
    AND la.neighbourhood_id = n.neighbourhood_id
    AND n.country_code = 'ES'
    GROUP BY l.host_id, h.host_name
    ) m
WHERE Rank <= 10
```

HOST_ID	HOST_NAME	NUMBER_OF_LISTINGS
4459553	Eva&Jacques	188
99018982	Apartamentos	95
32046323	Juan	88
28038703	Luxury Rentals Madrid	78
1391607	Aline	77

Query 10:
Find the top-10 rated (review_score_rating) apartments (id,name) in Barcelona.

Logic description:
Equi-join the Listings, Neighbourdhoods and Located_At tables. Filter to those listings that are apartments located in Barcelona and have a review scores rating. Rank listings in decreasing order of review scores ratings and select top 10 ranking rows.

SQL:

```
SELECT m.listing_id, m.listing_name, m.Rank
FROM
    (
    SELECT l.listing_id, l.listing_name, RANK() OVER (ORDER BY l.review_scores_rating DESC)
Rank
    FROM listings l, located_at la, neighbourhoods n
    WHERE l.listing_id = la.listing_id
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
    AND la.neighbourhood_id = n.neighbourhood_id
    AND n.city = 'Barcelona'
    AND l.listing_type = 'Apartment'
    AND l.review_scores_rating is not null
    ) m
WHERE m.Rank <= 10
```

LISTING_ID	LISTING_NAME	REVIEW_SCORES_RATING
71520	Charming apartment with fantastic views!	100
17468995	Private little room	100
11997102	Double Room - El Raval, Barcelona	100
337755	SEALONA VILA OLIMPICA BEACH	100
286105	Room at Gran Via Barcelona Spain	100

Query optimization

Although none of the queries above took a considerable amount of time to run, we optimized two queries' runtime by creating an index. We created an index on 'host_name' in the hosts table as follows:

```
CREATE INDEX host_index ON hosts(host_name)
```

As a result, this optimized the runtime of queries 4 and 5.

Query 4:
Without 'host_name' index:

Query Result ×
SQL | All Rows Fetched: 1 in 0.604 seconds

NUMBER_OF_LISTINGS
30393

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	1877
SORT		AGGREGATE	1	
HASH JOIN		RIGHT SEMI	42094	1877
Access Predicates				
L.HOST_ID=HOST_ID				
VIEW	SYS.VW_NSO_1		26659	1365
HASH JOIN		SEMI	26659	1365
Access Predicates				
H1.HOST_NAME=H2.HOST_NAME				
Filter Predicates				
H1.HOST_ID<>H2.HOST_ID				
TABLE ACCESS	HOSTS	FULL	26659	683
TABLE ACCESS	HOSTS	FULL	26659	683
TABLE ACCESS	LISTINGS	FULL	42094	512

With 'host_name' index:

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Query Result ×

SQL | All Rows Fetched: 1 in 0.088 seconds

	NUMBER_OF_LISTINGS
1	30393

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	813
SORT		AGGREGATE	1	
HASH JOIN		RIGHT SEMI	42094	813
Access Predicates				
L.HOST_ID=HOST_ID				
VIEW	SYS.VW_NSO_1		26659	301
HASH JOIN		SEMI	26659	301
Access Predicates				
H1.HOST_NAME=H2.HOST_NAME				
Filter Predicates				
H1.HOST_ID<>H2.HOST_ID				
VIEW	index$_join$_...		26659	150
HASH JOIN				
Access Predicates				
ROWID=ROWID				
INDEX	HOST_INDEX	FAST FULL S...	26659	88
INDEX	SYS_C0033598	FAST FULL S...	26659	100
VIEW	index$_join$_...		26659	150
HASH JOIN				
Access Predicates				
ROWID=ROWID				
INDEX	HOST_INDEX	FAST FULL S...	26659	88
INDEX	SYS_C0033598	FAST FULL S...	26659	100
TABLE ACCESS	LISTINGS	FULL	42094	512

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Query 5:

Without 'host_name' index:

Query Result ×

SQL | Fetched 50 rows in 0.052 seconds

	VIAJES_ECO_AVAILABLE_D...
1	09-NOV-18
2	17-NOV-18
3	24-DEC-18
4	22-JAN-19
5	02-FEB-19
6	05-FEB-19
7	17-FEB-19
8	18-FEB-19
9	15-NOV-18
10	21-NOV-18
11	22-NOV-18
12	26-NOV-18
13	11-DEC-18
14	21-DEC-18
15	30-DEC-18

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			367	1906
HASH		UNIQUE	367	1906
HASH JOIN			679	1905
Access Predicates				
A.LISTING_ID=L.LISTING_ID				
NESTED LOOPS			679	1905
NESTED LOOPS			990	1905
STATISTICS COLLECTOR				
HASH JOIN			5	1194
Access Predicates				
L.HOST_ID=H.HOST_ID				
TABLE ACCESS	HOSTS	FULL	3	683
Filter Predicates				
H.HOST_NAME='Viajes Eco'				
TABLE ACCESS	LISTINGS	FULL	42094	512
INDEX	SYS_C0033629	RANGE SCAN	198	2
Access Predicates				
A.LISTING_ID=L.LISTING_ID				
TABLE ACCESS	AVAILABILITY	BY INDEX RO...	145	194
TABLE ACCESS	AVAILABILITY	FULL	145	194

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

With 'host_name' index:

Query Result ×

SQL | Fetched 50 rows in 0.035 seconds

	VIAJES_ECO_AVAILABLE_DATES
1	09-NOV-18
2	17-NOV-18
3	24-DEC-18
4	22-JAN-19
5	02-FEB-19
6	05-FEB-19
7	17-FEB-19
8	18-FEB-19
9	15-NOV-18
10	21-NOV-18
11	22-NOV-18
12	26-NOV-18
13	11-DEC-18
14	21-DEC-18
15	30-DEC-18

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			367	1227
HASH		UNIQUE	367	1227
HASH JOIN			679	1226
Access Predicates				
A.LISTING_ID=L.LISTING_ID				
NESTED LOOPS			679	1226
NESTED LOOPS			990	1226
STATISTICS COLLECTOR				
HASH JOIN			5	516
Access Predicates				
L.HOST_ID=H.HOST_ID				
TABLE ACCESS	HOSTS	BY INDEX RO...	3	4
INDEX	HOST_INDEX	RANGE SCAN	3	1
Access Predicates				
H.HOST_NAME='Viajes Eco'				
TABLE ACCESS	LISTINGS	FULL	42094	512
INDEX	SYS_C0033629	RANGE SCAN	198	2
Access Predicates				
A.LISTING_ID=L.LISTING_ID				
TABLE ACCESS	AVAILABILITY	BY INDEX RO...	145	194
TABLE ACCESS	AVAILABILITY	FULL	145	194

Interface

Look at the Interface section in the deliverable 3

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Deliverable 3

Assumptions

The assumptions made follow on from the ones previously made. Any additional aspects taken into consideration are stated in the logic description of the relevant query.

Query Implementation

Query 1:

Description of logic:

Equi-join listings, located_at and neighbourhoods table to enrich each listing with its location data. Filter out those listings that have no square feet information (null) and those that have square feet set to 0 (as a listing cannot have 0 square feet, we considered this to be equivalent to undeclared). Group by city and count the number of distinct hosts. As a result, we have the count of hosts in each city that have declared the square footage for at least one of their properties.

SQL statement

```
SELECT n.city, COUNT(DISTINCT l.host_id) as NUMBER_OF_HOSTS
FROM listings l, located_at la, neighbourhoods n
WHERE l.listing_id = la.listing_id
AND la.neighbourhood_id = n.neighbourhood_id
AND l.square_feet is not null
AND l.square_feet <> 0
GROUP BY n.city
ORDER BY n.city
```

Query Result ×

SQL | All Rows Fetched: 3 in 0.041 seconds

	CITY	NUMBER_OF_HOSTS
1	Barcelona	227
2	Berlin	269
3	Madrid	150

Query 2:

Description of logic:

On one hand, count the number of listings per neighbourhood in Madrid. This is done by joining the neighbourhoods, located_at and listings tables, filtering to those listings in Madrid, and grouping by neighbourhood_name and neighbourhood_id while counting the number of listings for each group. On the other hand, get the median listing for each neighbourhood. This done by verifying whether the listing_id for its corresponding combination of review_scores_rating and neighbourhood_id is in the singleton list representing

the median for that given neighbourhood. The median for a given neighbourhood is obtained as follows: select all of its listings with non-null review_scores_rating, order them by decreasing order and assign a distinct rank to each row (smallest to highest score, largest to lowest); join this table with a single row representing half the count of the aforementioned list of listings for the given neighbourhood (position of median row); filter to that row where the distinct rank is equal to the position of the median row. Finally, join the first table containing the counts of listings per neighbourhood of Madrid with the table containing the median review_scores_rating of each neighbourhood. Rank the resulting table over review_scores_rating and count of listings, both in decreasing order, and select top 5 ranking neighbourhoods.

SQL statement

```
SELECT * FROM
    (
    SELECT l1.neighbourhood_name,  RANK() OVER (ORDER BY m2.review_scores_rating DESC,
l1.number_of_listings DESC) Rank
    FROM
        (
        SELECT n.neighbourhood_name, n.neighbourhood_id ,COUNT(*) as number_of_listings
        FROM neighbourhoods n, located_at la, listings l
        WHERE n.neighbourhood_id = la.neighbourhood_id
        AND la.listing_id = l.listing_id
        AND n.city = 'Madrid'
        GROUP BY n.neighbourhood_name, n.neighbourhood_id
        ) l1,
        (
        SELECT l2.review_scores_rating, la2.neighbourhood_id
        FROM located_at la2, listings l2
        WHERE la2.listing_id = l2.listing_id
        AND l2.listing_id IN
            (
            SELECT x3.listing_id FROM
                (
                SELECT x1.*, x2.median_row, rownum r
                FROM
                    (
                    SELECT l3.listing_id
                    FROM listings l3, located_at la3
                    WHERE l3.listing_id = la3.listing_id
                    AND la3.neighbourhood_id = la2.neighbourhood_id
                    AND l3.review_scores_rating is not null
                    ORDER BY l3.review_scores_rating desc
                    ) x1,
                    (
                    SELECT ROUND(COUNT(*)/2) as median_row
```

```
                    FROM
                        (
                        SELECT l4.listing_id
                        FROM listings l4, located_at la4
                        WHERE l4.listing_id = la4.listing_id
                        AND la4.neighbourhood_id = la2.neighbourhood_id
                        AND l4.review_scores_rating is not null
                        ORDER BY l4.review_scores_rating desc
                        )
                    ) x2
                ) x3
            WHERE x3.r = x3.median_row
            )
        ) m2
    WHERE l1.neighbourhood_id = m2.neighbourhood_id
    )
WHERE Rank <= 5
```

Query Result

SQL | All Rows Fetched: 5 in 40.964 seconds

	NEIGHBOURHOOD_NAME	RANK
1	Estrella	1
2	Tetuán	2
3	Hispanoamérica	3
4	Vallehermosa	4
5	Vicálvaro	4

Query 3:

Description of logic:

Count the number of listings per hosts by equi-joining the hosts and listings tables, and grouping by host_id and host_name while counting the number of listings. Rank all of the hosts over the count of listings per host in decreasing order and select the rows of rank 1.

SQL statement

```
SELECT host_id, host_name
FROM
    (
    SELECT m.host_id, m.host_name, RANK() OVER (ORDER BY number_of_listings DESC) Rank
    FROM
        (
        SELECT h.host_id, h.host_name, COUNT(*) as number_of_listings
        FROM listings l, hosts h
        WHERE l.host_id = h.host_id
```

```
        GROUP BY h.host_id, h.host_name
        ) m
    )
WHERE Rank = 1
```

Query Result

SQL | All Rows Fetched: 1 in 0.162 seconds

	HOST_ID	HOST_NAME
1	4459553	Eva&Jacques

Query 4:

Description of logic:

Equi-join listings, located_at, neighbourhoods, availability, verified_through and verification_channels tables in order to obtain all the relevant information. Apply the relevant filtering: Berlin apartments with flexible cancellation policies that have a review_scores_location of at least 8, listed by a host with a verifiable government id, with at least 2 beds and that are available at least one day between 01-03-2019 and 30-04-2019 (inclusive). We assumed that a verifiable government id is only equivalent to the verification channel named 'government_id' and did not include any other channels. Group by the resulting filtered data on listing_id and average the price over the availability period for those days on which the listing is available for each listing_id (no additional filtering was required here as our availability table only contains available days and not non-available days). Rank the resulting relation on the average price in ascending order and select the top 5 ranking rows.

SQL statement

```
SELECT m2.listing_id, m2.avg_price FROM
    (
    SELECT m1.*, RANK() OVER (ORDER BY avg_price) Rank FROM
        (
        SELECT l.listing_id, ROUND(AVG(a.price),2) as avg_price
        FROM listings l, located_at la, neighbourhoods n, availability a, verified_through
vt, verification_channels vc
        WHERE l.listing_id = la.listing_id
        AND la.neighbourhood_id = n.neighbourhood_id
        AND a.listing_id = l.listing_id
        AND vt.host_id = l.host_id
        AND vc.verification_channel_id = vt.verification_channel_id
        AND n.city = 'Berlin'
        AND l.listing_type = 'Apartment'
        AND a.date_ >= to_date('01-MAR-19','DD-MON-YY')
        AND a.date_ <= to_date('30-APR-19','DD-MON-YY')
        AND l.beds >= 2
        AND l.review_scores_location >= 8
        AND l.cancellation_policy = 'flexible'
```

```
        AND vc.verification_channel_name = 'government_id'
        GROUP BY l.listing_id
        ) m1
    ) m2
WHERE m2.Rank <= 5
```

Query Result

SQL | All Rows Fetched: 5 in 0.59 seconds

	LISTING_ID	AVG_PRICE
1	1490274	20
2	24043706	21.07
3	1368460	21.29
4	7071541	22
5	6691656	22

Query 5:

Description of logic:

Equi-join the listings, includes and amenities tables in order to obtain a list of listings alongside with all of the amenities available in each listings. Filter to Wifi, Internet, TV and Free street parking amenities and to those listings that have a review_scores_rating. Group by listing_id, accommodates (number of people a listing accommodates) and review_scores_rating keeping those groups that have at least 2 rows. This is equivalent to keeping those listings that have at least two of the previously listed amenities. Considering the listings in groups based on the number of people they can accommodate, rank the listings within each of these groups over review_scores_rating in descending order. Finally, select all those listings that rank in top 5 in their respective accommodates group.

SQL statement

```
SELECT * FROM
    (
    SELECT m.accommodates, m.listing_id,
    RANK() OVER (PARTITION BY m.accommodates ORDER BY m.review_scores_rating DESC) Rank
    FROM
        (
        SELECT l.accommodates, l.listing_id, l.review_scores_rating
        FROM listings l, includes i, amenities a
        WHERE i.listing_id = l.listing_id
        AND i.amenity_id = a.amenity_id
        AND a.amenity_name IN ('Wifi', 'Internet', 'TV', 'Free street parking')
        AND l.review_scores_rating is not null
        GROUP BY l.listing_id, l.accommodates, l.review_scores_rating
        HAVING COUNT(*) > 1
        ) m
    )
```

```
WHERE RANK <= 5
```

Query Result

SQL | Fetched 50 rows in 0.172 seconds

	ACCOMMOD...	LISTING_ID	RANK
1	1	109369	1
2	1	179488	1
3	1	240735	1
4	1	250121	1
5	1	287660	1
6	1	337292	1
7	1	395063	1
8	1	456165	1
9	1	510973	1
10	1	545037	1

Query 6:

Description of logic:

Equi-join listings and reviewed tables and group by host_id and listing_id while counting the number of reviews for each listing. For each host in the resulting relation, rank the listings over the count of reviews in decreasing order. Finally, select the host_id and listing_id combinations where the listing ranked in the top 3 among the listings hosted by the same host. Note that we ordered by host_id in order to ensure we keep adjacent the listings for a given host and we included the count of reviews and ranking data in the output in order to provide more insight.

SQL statement

```
SELECT * FROM
    (
    SELECT m1.host_id, m1.listing_id, m1.popularity, RANK() OVER (PARTITION BY m1.host_id
ORDER BY popularity DESC) Rank
        FROM
        (
        SELECT l.host_id, l.listing_id, COUNT(*) as popularity
        FROM listings l, reviewed r
        WHERE l.listing_id = r.listing_id
        GROUP BY l.host_id, l.listing_id
        ) m1
    ) m2
WHERE m2.rank <= 3
ORDER BY m2.host_id
```

Query Result

SQL | Fetched 50 rows in 0.954 seconds

	HOST_ID	LISTIN...	POPULARITY	RANK
1	2217	2015	118	1
2	2217	21315310	50	2
3	2217	18773184	36	3
4	3073	6287375	21	1
5	3718	3176	143	1
6	4108	3309	25	1
7	5154	18132872	48	1
8	10704	8217664	57	1
9	10704	733941	34	2
10	10704	9572534	15	3
11	10966	28743771	3	1
12	10966	29664434	1	2
13	11015	1247590	276	1
14	11780	801290	7	1
15	12360	21038831	2	1

Query 7:

Description of logic:

Equi-join the listings, located_at, neighbourhoods, includes and amenities tables. Filter to those rows where the listing is located in Berlin and is a room of type Private room. Group by the resulting relation by neighbourhood_name and amenity_name while counting the number of listings for each combination. For each neighbourhood in the resulting relation, rank the amenities that are available there on the count of times they appear in listings in that given neighbourhood in decreasing order. Select those combinations of neighbourhood_name and amenity_name where the amenity ranked top 3 in the given neighbourhood. Note, similarly to the previous query, we included the count of listings and rank in the output of the query for informational purposes.

SQL statement

```
SELECT * FROM
    (
    SELECT m.neighbourhood_name, m.amenity_name, m.popularity, RANK() OVER (PARTITION BY
m.neighbourhood_name ORDER BY m.popularity DESC) Rank
    FROM
        (
        SELECT n.neighbourhood_name, a.amenity_name, COUNT(*) as popularity
        FROM listings l, located_at la, neighbourhoods n, includes i, amenities a
        WHERE l.listing_id = la.listing_id
        AND la.neighbourhood_id = n.neighbourhood_id
        AND l.listing_id = i.listing_id
        AND i.amenity_id = a.amenity_id
        AND n.city = 'Berlin'
        AND l.room_type = 'Private room'
        GROUP BY n.neighbourhood_name, a.amenity_name
```

```
        ) m
    )
WHERE Rank <= 3
```

Query Result

SQL | Fetched 50 rows in 0.122 seconds

	NEIGHBOURHOOD_NAME	AMENITY_NAME	POPULARITY	RANK
1	Adlershof	Essentials	7	1
2	Adlershof	Wifi	6	2
3	Adlershof	Heating	6	2
4	Alt-Hohenschönhausen	Heating	17	1
5	Alt-Hohenschönhausen	Essentials	16	2
6	Alt-Hohenschönhausen	Wifi	16	2
7	Alt-Treptow	Wifi	63	1
8	Alt-Treptow	Heating	60	2
9	Alt-Treptow	Essentials	60	2
10	Altglienicke	Kitchen	9	1

Query 8:

Description of logic:

One one hand, filter the listings table to those listings that have a review_scores_communication and subsequently group by host_id while averaging the review_scores_communication for each. On the other hand, equi-join the hosts and verified_through tables to obtain the verification_channel_id for each host. Group by host_id while counting the number of distinct verification_channel_id's for each. Join both of these resulting relations on host_id and order by the hosts based on the count of verification_channel_id's in decreasing order. Select the first row in resulting table. Repeat the aforementioned computation but sort the count of verification_channel_id's in ascending order instead. Finally, we substract the average communication scores of the first host with the second. Note: the hosts with the least diverse way of verifications (0) did not host a listings with a review_scores_communication, hence these a filtered out by the 'is not null' condition and we determined that this is the correct approach as an average of null values does not make sense.

SQL statement

```
SELECT maxh.avg_comm_score - minh.avg_comm_score as difference
FROM
    (
    SELECT * FROM
        (
        SELECT m1.host_id, m1.avg_comm_score, m2.diversity
        FROM
            (
            SELECT l.host_id, avg(l.review_scores_communication) as avg_comm_score
            FROM listings l
            WHERE l.review_scores_communication is not null
            GROUP BY l.host_id
            ) m1,
            (
            SELECT h.host_id, COUNT(DISTINCT vt.verification_channel_id) as diversity
```

```
            FROM hosts h, verified_through vt
            WHERE vt.host_id = h.host_id
            GROUP BY h.host_id
            ) m2
        WHERE m1.host_id = m2.host_id
        ORDER BY m2.diversity desc
        )
    WHERE ROWNUM = 1
    ) maxh,
    (
    SELECT * FROM
        (
        SELECT m1.host_id, m1.avg_comm_score, m2.diversity
        FROM
            (
            SELECT l.host_id, avg(l.review_scores_communication) as avg_comm_score
            FROM listings l
            WHERE l.review_scores_communication is not null
            GROUP BY l.host_id
            ) m1,
            (
            SELECT h.host_id, COUNT(DISTINCT vt.verification_channel_id) as diversity
            FROM hosts h
            LEFT OUTER JOIN verified_through vt ON vt.host_id = h.host_id
            GROUP BY h.host_id
            ) m2
        WHERE m1.host_id = m2.host_id
        ORDER BY m2.diversity asc
        )
    WHERE ROWNUM = 1
    ) minh
```

Query Result ×

SQL | All Rows Fetched: 1 in 0.144 seconds

	DIFFERENCE
1	0

Query 9:

Description of logic:

Equi-join the listings, located_at and neighbourhoods tables to obtain location data for each listing. Filter the resulting listings to those that have a room type with an average of accommodates greater than 3. Equi-join the

resulting relation with the reviewed table to get all reviews per listings. Subsequently group by city while counting the number of reviews. Rank the resulting relation over the count of reviews in decreasing order and select top ranking city.

SQL statement

```
SELECT city FROM
    (
    SELECT m2.*, RANK() OVER (ORDER BY number_of_reviews DESC) Rank
    FROM
        (
        SELECT m1.city, COUNT(*) as number_of_reviews
        FROM
            (
            SELECT l.listing_id, n.city
            FROM listings l, located_at la, neighbourhoods n
            WHERE l.listing_id = la.listing_id
            AND la.neighbourhood_id = n.neighbourhood_id
            AND l.room_type IN
                (
                SELECT l2.room_type
                FROM listings l2
                GROUP BY l2.room_type
                HAVING AVG(l2.accommodates) > 3
                )
            ) m1,
            reviewed re
        WHERE re.listing_id = m1.listing_id
        GROUP BY m1.city
        ) m2
    )
WHERE RANK = 1
```

Query Result

SQL | All Rows Fetched: 1 in 0.349 seconds

	CITY
1	Madrid

Query 10:

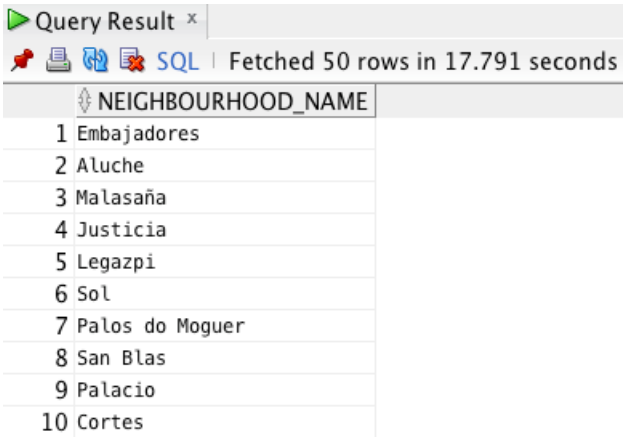
Description of logic:

Filter the neighbourhoods table to those in Madrid. For each of neighbourhood count both the number of listings that were occupied at some point in 2019 and for which the host joined prior to 01.06.2017 and the total number of listings. For the former, we count the number of days a listing is available in 2019 and keep it if

this number is less than the number of days in 2019 in the days table. Keep those neighbourhoods that have at least 50% of their listings occupied in 2019 and for which the host joined prior to 01.06.2017.

SQL statement

```
SELECT n.neighbourhood_name
FROM neighbourhoods n
WHERE n.city = 'Madrid'
AND EXISTS (
    SELECT * FROM
        (
        SELECT COUNT(*) as occupied
        FROM
            (
            SELECT l1.listing_id, COUNT(*) as days_free_in_19
            FROM listings l1, located_at la1, hosts h1, availability a1
            WHERE l1.listing_id = la1.listing_id
            AND la1.neighbourhood_id = n.neighbourhood_id
            AND h1.host_id = l1.host_id
            AND l1.listing_id = a1.listing_id
            AND h1.host_since < to_date('01-JUN-17','DD-MON-YY')
            AND a1.date_ > to_date('31-DEC-18','DD-MON-YY')
            GROUP BY l1.listing_id
            ) m1
        WHERE m1.days_free_in_19 < (SELECT COUNT (DISTINCT DATE_) as days_in_19 FROM DAYS
WHERE DATE_ > to_date('31-DEC-18','DD-MON-YY'))
        ) x1,
        (
        SELECT COUNT(*) as total
        FROM listings l2, located_at la2
        WHERE l2.listing_id = la2.listing_id
        AND la2.neighbourhood_id = n.neighbourhood_id
        ) x2
    WHERE x1.occupied/x2.total >= 0.5
    )
```



Query Result | SQL | Fetched 50 rows in 17.791 seconds

	NEIGHBOURHOOD_NAME
1	Embajadores
2	Aluche
3	Malasaña
4	Justicia
5	Legazpi
6	Sol
7	Palos do Moguer
8	San Blas
9	Palacio
10	Cortes

Query 11:

Description of logic:

On one hand equi-join the availability, located_at and neighbourhoods tables. Filter to those rows where the listing was available in 2018. Group by country while counting the number of distinct listings. This gives us the number of listings that were available at some date in 2018 per country. One the other hand, equi-join the located_at and neighbourhoods tables and group by country while counting the number of listings per country. Equi-join the two resulting relations and filter to those countries where the number of listings that were available in 2018 represent at least 20% of the total listings in that country.

SQL statement

```
SELECT l1.country
FROM
    (
    SELECT n.country, COUNT(DISTINCT a.listing_id) as available_listings
    FROM availability a, located_at la, neighbourhoods n
    WHERE a.listing_id = la.listing_id
    AND la.neighbourhood_id = n.neighbourhood_id
    AND a.date_ < to_date('01-JAN-19','DD-MON-YY')
    GROUP BY n.country
    ) l1,
    (
    SELECT n1.country, COUNT(DISTINCT la1.listing_id) as total_listings
    FROM located_at la1, neighbourhoods n1
    WHERE la1.neighbourhood_id = n1.neighbourhood_id
    GROUP BY n1.country
    ) l2
WHERE l1.country = l2.country
AND l1.available_listings / l2.total_listings >= 0.2
```

Query 12:

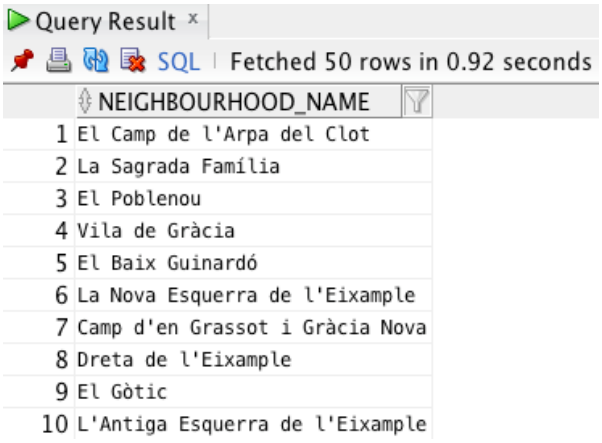
Description of logic:

For each neighbourhood of Barcelona, on one hand count the number of listings with cancellation policies that are strict with grace period and on the other hand count the total number of listings. Verify that the ratio of the resulting counts is greater than 5% and if it is return the neighbourhood_name. As a result, we obtained the required list of neighbourhoods.

SQL statement

```
SELECT n.neighbourhood_name
FROM neighbourhoods n
WHERE n.city = 'Barcelona'
AND EXISTS (
    SELECT * FROM
        (
        SELECT m1.strict_w_grace/m2.no_of_listings as ratio
        FROM
            (
            SELECT COUNT(*) as strict_w_grace
            FROM listings l2, located_at la2
            WHERE l2.listing_id = la2.listing_id
            AND la2.neighbourhood_id = n.neighbourhood_id
            AND l2.cancellation_policy = 'strict_14_with_grace_period'
            ) m1,
            (
            SELECT COUNT(*) as no_of_listings
            FROM listings l3, located_at la3
            WHERE l3.listing_id = la3.listing_id
            AND la3.neighbourhood_id = n.neighbourhood_id
            ) m2
        ) m3
    WHERE m3.ratio > 0.05)
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/


Query Result ×

SQL | Fetched 50 rows in 0.92 seconds

	NEIGHBOURHOOD_NAME
1	El Camp de l'Arpa del Clot
2	La Sagrada Família
3	El Poblenou
4	Vila de Gràcia
5	El Baix Guinardó
6	La Nova Esquerra de l'Eixample
7	Camp d'en Grassot i Gràcia Nova
8	Dreta de l'Eixample
9	El Gòtic
10	L'Antiga Esquerra de l'Eixample

Query Analysis

Selected Queries (and why)

We decided to optimize queries 6, 7 and 11 through the creation of additional indices. Although these queries are not the longest running ones, these were selected due to the potential for optimization. As a result of the fact that Oracle SQL Developer automatically creates indices for Primary Keys and Unique columns, this left us with the ability to optimize those queries where there is a join or filtering on a column that is neither a Primary Key or one that has Unique constraint. We created an index on a join column that is a Foreign Key in the respective table for query 6. For query 7 and 11, we created an index on a column that is used for filtering.

Query 6

Create index SQL:

```
CREATE INDEX listing_review_index ON reviewed(listing_id)
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Initial Running time (left) / Optimized Running time(right):

Query Result

SQL | Fetched 50 rows in 0.954 seconds

	HOST_ID	LISTIN...	POPULARITY	RANK
1	2217	2015	118	1
2	2217	21315310	50	2
3	2217	18773184	36	3
4	3073	6287375	21	1
5	3718	3176	143	1
6	4108	3309	25	1
7	5154	18132872	48	1
8	10704	8217664	57	1
9	10704	733941	34	2
10	10704	9572534	15	3
11	10966	28743771	3	1
12	10966	29664434	1	2
13	11015	1247590	276	1
14	11780	801290	7	1
15	12360	21038831	2	1

Query Result

SQL | Fetched 50 rows in 0.57 seconds

	HOST_ID	LISTING_ID	POPULARITY	RANK
1	2217	2015	118	1
2	2217	21315310	50	2
3	2217	18773184	36	3
4	3073	6287375	21	1
5	3718	3176	143	1
6	4108	3309	25	1
7	5154	18132872	48	1
8	10704	8217664	57	1
9	10704	733941	34	2
10	10704	9572534	15	3
11	10966	28743771	3	1
12	10966	29664434	1	2
13	11015	1247590	276	1
14	11780	801290	7	1
15	12360	21038831	2	1

Explain the improvement:
By creating an index on listing_id within the reviewed table, when performing the join between the listings and reviewed relation on listing_id, it allows to perform a range scan and results in loading less pages of the reviewed relation into memory.

Initial plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1272377	16645
VIEW			1272377	16645
Filter Predicates				
M2.RANK<=3				
WINDOW		SORT PUSHE...	1272377	16645
Filter Predicates				
RANK() OVER (PARTITION BY L.HOST_ID ORDER BY COUNT(*) DESC)<=3				
HASH		GROUP BY	1272377	16645
HASH JOIN			1272377	1681
Access Predicates				
L.LISTING_ID=R.LISTING_ID				
TABLE ACCESS	LISTINGS	FULL	42094	512
TABLE ACCESS	REVIEWED	FULL	1272377	1166

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Improved plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1272377	16294
VIEW			1272377	16294
Filter Predicates				
M2.RANK<=3				
WINDOW		SORT PUSHE...	1272377	16294
Filter Predicates				
RANK() OVER (PARTITION BY L.HOST_ID ORDER BY COUNT(*) DESC)<=3				
HASH		GROUP BY	1272377	16294
HASH JOIN			1272377	1330
Access Predicates				
L.LISTING_ID=R.LISTING_ID				
NESTED LOOPS			1272377	1330
STATISTICS COLLECT				
TABLE ACCESS	LISTINGS	FULL	42094	512
INDEX	LISTING_REVIE...	RANGE SCAN	30	815
Access Predicates				
L.LISTING_ID=R.LISTING_ID				
INDEX	LISTING_REVIE...	FAST FULL S...	1272377	815

Query 7

Create index SQL:

```
CREATE INDEX room_type_index ON listings(room_type)
```

Initial Running time:

Query Result

SQL | Fetched 50 rows in 0.122 seconds

	NEIGHBOURHOOD_NAME	AMENITY_NAME	POPULARITY	RANK
1	Adlershof	Essentials	7	1
2	Adlershof	Wifi	6	2
3	Adlershof	Heating	6	2
4	Alt-Hohenschönhausen	Heating	17	1
5	Alt-Hohenschönhausen	Essentials	16	2
6	Alt-Hohenschönhausen	Wifi	16	2
7	Alt-Treptow	Wifi	63	1
8	Alt-Treptow	Heating	60	2
9	Alt-Treptow	Essentials	60	2
10	Altglienicke	Kitchen	9	1

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Optimized Running time:

Query Result

SQL | Fetched 50 rows in 0.104 seconds

	NEIGHBOURHOOD_NAME	AMENITY_NAME	POPULARITY	RANK
1	Adlershof	Essentials	7	1
2	Adlershof	Wifi	6	2
3	Adlershof	Heating	6	2
4	Alt-Hohenschönhausen	Heating	17	1
5	Alt-Hohenschönhausen	Essentials	16	2
6	Alt-Hohenschönhausen	Wifi	16	2
7	Alt-Treptow	Wifi	63	1
8	Alt-Treptow	Heating	60	2
9	Alt-Treptow	Essentials	60	2
10	Altglienicke	Kitchen	9	1

Explain the improvement:
By creating an index on the room_type column of the listings table, when the query extracts the listings from disk it can now extract only those pages that include listings of type ‘Private room’ instead of doing a full scan.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Initial plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
Filter Predicates				
RANK<=3				
WINDOW		SORT PUSHE...	11647	3283
Filter Predicates				
RANK() OVER (PARTITION BY N.NEIGHBOURHOOD_NAME ORDER BY COUNT(*) DESC)				
HASH		GROUP BY	11647	3283
HASH JOIN			148357	992
Access Predicates				
I.AMENITY_ID=A.AMENITY_ID				
VIEW	index$_join$_...		181	2
HASH JOIN				
Access Predicates				
ROWID=ROWID				
INDEX	SYS_C0017323	FAST FULL S...	181	1
INDEX	SYS_C0017324	FAST FULL S...	181	1
HASH JOIN			148357	989
Access Predicates				
L.LISTING_ID=I.LISTING_ID				
HASH JOIN			7547	540
Access Predicates				
L.LISTING_ID=LA.LISTING_ID				
TABLE ACCESS	LISTINGS	FULL	18578	512
Filter Predicates				
L.ROOM_TYPE='Private room'				
HASH JOIN			17101	28
Access Predicates				
LA.NEIGHBOURHOOD_ID=N.NEIGHBOURHOOD_ID				
TABLE ACCESS	NEIGHBOURHO...	FULL	91	3
Filter Predicates				
N.CITY='Berlin'				
TABLE ACCESS	LOCATED_AT	FULL	42094	25
TABLE ACCESS	INCLUDES	FULL	822131	447

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Improved plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
HASH		GROUP BY	11647	2965
HASH JOIN			148357	673
Access Predicates				
I.AMENITY_ID=A.AMENITY_ID				
VIEW	index$_join$_...		181	2
HASH JOIN				
Access Predicates				
ROWID=ROWID				
INDEX	SYS_C0017323	FAST FULL S...	181	1
INDEX	SYS_C0017324	FAST FULL S...	181	1
HASH JOIN			148357	671
Access Predicates				
L.LISTING_ID=I.LISTING_ID				
HASH JOIN			7547	222
Access Predicates				
L.LISTING_ID=LA.LISTING_ID				
VIEW	index$_join$_...		18578	194
Filter Predicates				
L.ROOM_TYPE='Private room'				
HASH JOIN				
Access Predicates				
ROWID=ROWID				
INDEX	ROOM_TYPE_I...	RANGE SCAN	18578	67
Access Predicates				
L.ROOM_TYPE='Private room'				
INDEX	SYS_C0033617	FAST FULL S...	18578	158
HASH JOIN			17101	28
Access Predicates				
LA.NEIGHBOURHOOD_ID=N.NEIGHBOURHOOD_ID				
TABLE ACCESS	NEIGHBOURHO...	FULL	91	3
Filter Predicates				
N.CITY='Berlin'				
TABLE ACCESS	LOCATED_AT	FULL	42094	25
TABLE ACCESS	INCLUDES	FULL	822131	447

Query 11

Create index SQL:

```
CREATE INDEX date_index ON availability(date_)
```

Initial Running time:

Query Result

SQL | All Rows Fetched: 2 in 0.407 seconds

	COUNTRY
1	Spain
2	Germany

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Optimized Running time:

Query Result × | Explain Plan ×

SQL | All Rows Fetched: 2 in 0.223 seconds

	COUNTRY
1	Spain
2	Germany

Explain the improvement:
Similarly, by creating an index on date_ in the availability table it allows for the query executor to avoid doing a full scan of the table and rather do a range scan by leveraging the index which results in a lower number of pages being loaded from the database.

Initial plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	6506
HASH JOIN			1	6506
Access Predicates				
L1.COUNTRY=L2.COUNTRY				
Filter Predicates				
L1.AVAILABLE_LISTINGS/L2.TOTAL_LISTINGS>=0.2				
VIEW			2	6476
HASH		GROUP BY	2	6476
VIEW	SYS.VM_NWVW_1		30700	6476
HASH		GROUP BY	30700	6476
HASH JOIN			30700	6194
Access Predicates				
LA.NEIGHBOURHOOD_ID=N.NEIGHBOURHOOD_ID				
TABLE ACCESS	NEIGHBOURHO...	FULL	224	3
HASH JOIN		SEMI	30700	6190
Access Predicates				
A.LISTING_ID=LA.LISTING_ID				
TABLE ACCESS	LOCATED_AT	FULL	42094	25
TABLE ACCESS	AVAILABILITY	FULL	915686	6163
Filter Predicates				
A.DATE_<TO_DATE('01-JAN-19','DD-MON-YY')				
VIEW			2	30
HASH		GROUP BY	2	30
HASH JOIN			42094	28
Access Predicates				
LA1.NEIGHBOURHOOD_ID=N1.NEIGHBOURHOOD_ID				
TABLE ACCESS	NEIGHBOURHO...	FULL	224	3
TABLE ACCESS	LOCATED_AT	FULL	42094	25

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

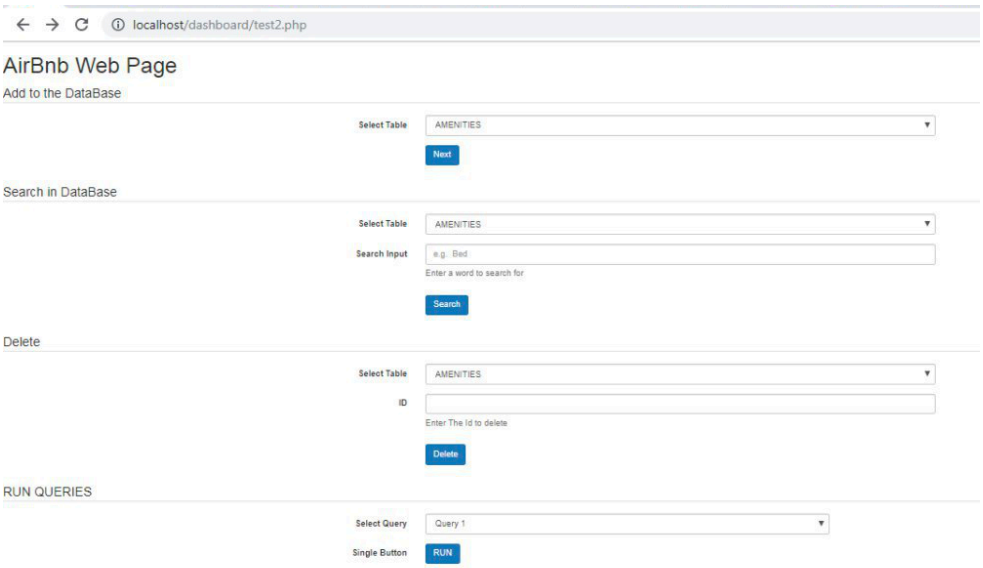
Improved plan

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	6115
HASH JOIN			1	6115
Access Predicates: L1.COUNTRY=L2.COUNTRY				
Filter Predicates: L1.AVAILABLE_LISTINGS/L2.TOTAL_LISTINGS>=0.2				
VIEW			2	6085
HASH		GROUP BY	2	6085
VIEW	SYS.VM_NWVW_1		30700	6085
HASH		GROUP BY	30700	6085
HASH JOIN			30700	5802
Access Predicates: LA.NEIGHBOURHOOD_ID=N.NEIGHBOURHOOD_ID				
TABLE ACCESS	NEIGHBOURHO...	FULL	224	3
HASH JOIN		SEMI	30700	5799
Access Predicates: A.LISTING_ID=LA.LISTING_ID				
TABLE ACCESS	LOCATED_AT	FULL	42094	25
TABLE ACCESS	AVAILABILITY	BY INDEX RO...	915686	5771
INDEX	DATE_INDEX	RANGE SCAN	915686	2437
Access Predicates: A.DATE_<TO_DATE('01-JAN-19','DD-MON-YY')				
VIEW			2	30
HASH		GROUP BY	2	30
HASH JOIN			42094	28
Access Predicates: LA1.NEIGHBOURHOOD_ID=N1.NEIGHBOURHOOD_ID				
TABLE ACCESS	NEIGHBOURHO...	FULL	224	3
TABLE ACCESS	LOCATED_AT	FULL	42094	25

Interface

We have implemented the web user interface using PHP

In the following figures you can see our implementation of GUI for the Airbnb database.
Here is the main page

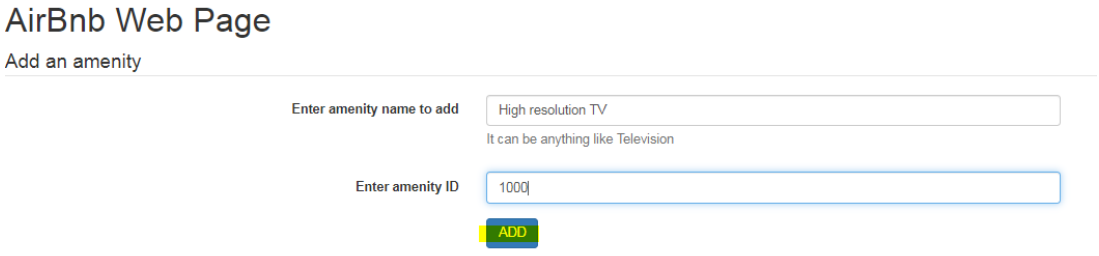


Our implementation does the following tasks:

- In first section, one can select a table to add a row to it
- in the second section, one can search in different tables
- in the third section, one can select a table and delete a row form it
- in the fourth section, one can select a query from the 12 queries and run it

In the following figures it is shown how this works:

- ADD: First we select the table that we want to add a row to it, then by pressing “Next” button we go to the following page where we can enter the needed information
  (To add an amenity we enter AMENITY_NAME , AMENITY_ID and then we press ADD button)



And here is the result which shows all amenities as well as added one

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

trying to add amenity_id: 1000 and amenity_name :High resolution TV

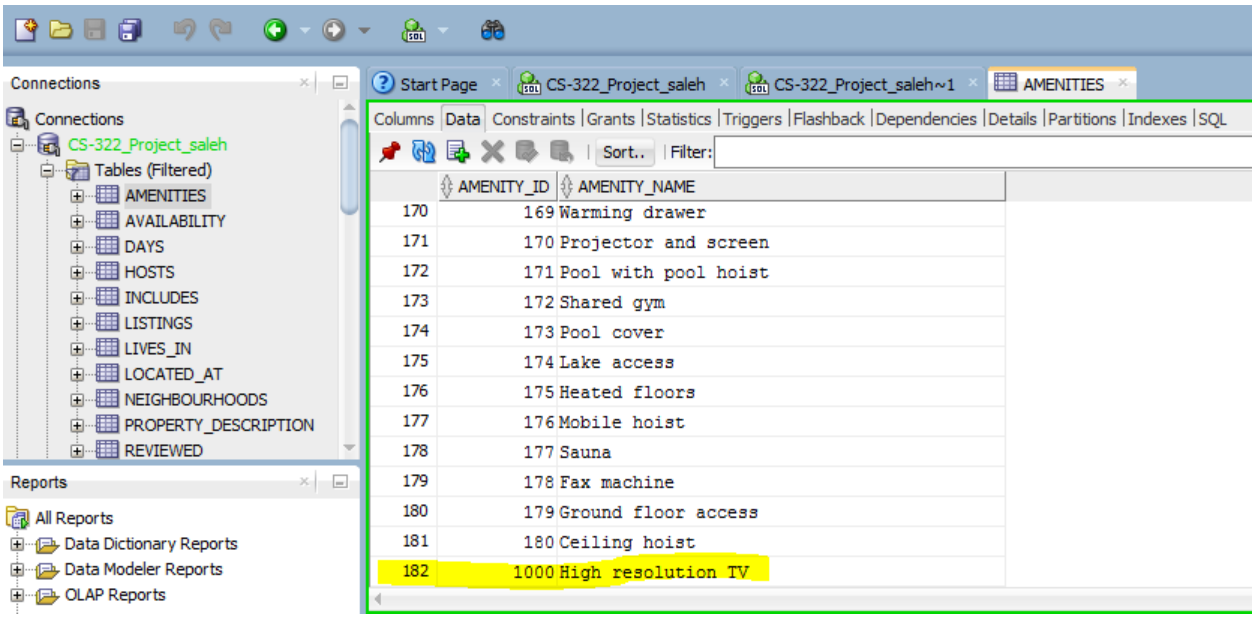
75	toilet"
54	24-hour check-in
68	Accessible-height bed
69	Accessible-height toilet
3	Air conditioning
151	Air purifier
168	Amazon Echo
129	BBQ grill
104	Baby bath
114	Baby monitor
84	Babysitter recommendations
119	Balcony
86	Bath towel
94	Bathroom essentials
71	Bathtub
111	Bathtub with bath chair

.
.
.

74	Wide clearance to shower
41	Wide doorway
44	Wide entryway
73	Wide hallway clearance
2	Wifi
97	Window guards
160	Wine cooler
60	translation missing: en.hosting_amenity_49
48	translation missing: en.hosting_amenity_50

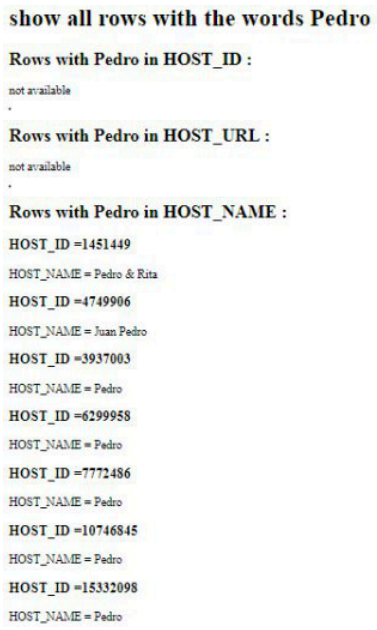
Return To main Page

And here is the table in the database



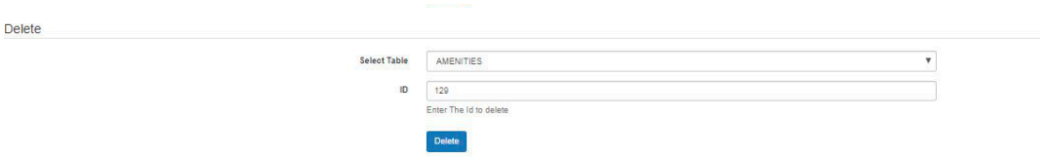
- Search: We first select the table that we want to search on, then to search for a keyword we write it and press the search button

and the result will be shown in the result page



here for example:
there is no match for "Pedro" in HOST_ID or HOST_URL columns but there are many in the HOST_NAME

- Delete: we first select the table that we want to delet a row from it and then select the ID of that row



- Run Queries: In this section, one can select a query among the 12 queries and run it

the result of the selected query would appear in the next page:
as an example, here is the result of the query 1:

Results of Query 1

CITY	NUMBER_OF_HOSTS
Barcelona	227
Berlin	269
Madrid	150

Return To main Page

General Comments

No comments!