Databases Project – Spring 2019

# Name: Yong Hao, Zeng Yanxi, Zhang Yuehan Team No: 49

Contents

Contents 1

Deliverable 1 2

Assumptions 2

Entity Relationship Schema 2

Schema 2

Description 2

Relational Schema 2

ER schema to Relational schema 2

DDL 3

General Comments 3

Deliverable 2 4

Assumptions 4

Data Loading 4

Query Implementation 4

Query a: 4

Description of logic: 4

SQL statement 4

Interface 4

Design logic Description 4

Screenshots 4

General Comments 4

Deliverable 3 5

Assumptions 5

Query Implementation 5

Query a: 5

Description of logic: 5

SQL statement 5

Query Analysis 5

Selected Queries (and why) 5

Query 1 5

Query 2 5

Query 3 5

Interface 6

Design logic Description 6

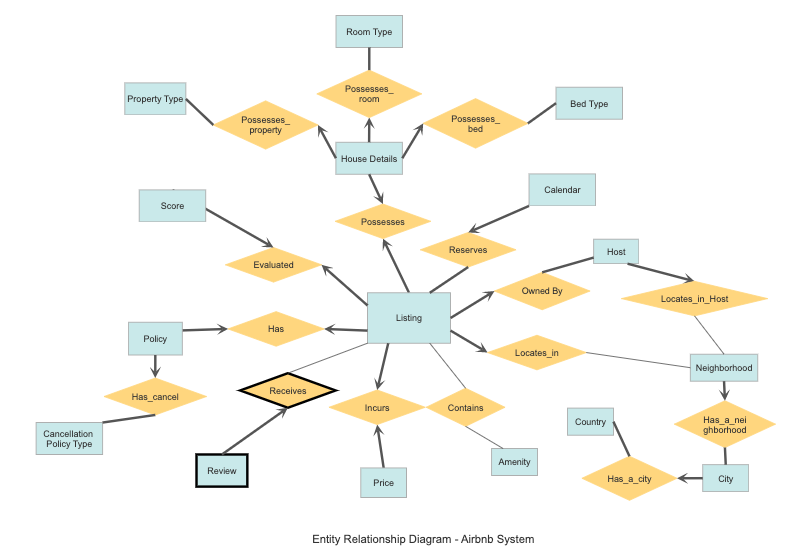
Screenshots 6

General Comments 6

# Deliverable 1

## Entity Relationship Schema

### Schema



### Description

<Describe all the choices you made for Entities and Relationships>

Justification of the design choices & Description of the data constraints

**Entities:**

* Main entities:

Listing – listing items contain the house records

Host – contains the host information who provide the listings in the Airbnb System

Calendar – contains the time properties that related to each listing item

Price – contains the price information under different circumstance for each listing items

* Entities generated from categorical variables:

Benefit: main table can only store an integer instead of a longer string, makes it easier to control what someone can insert in that field (with referential integrity constraint).

Neighbourhood – original categorical variable(attribute) of entity *Listing* and *Host*.

City – original categorical variable(attribute) of entity *Listing.*

Country – original categorical variable(attribute) of entity *Listing.*

Bed Type – original categorical variable(attribute) of entity *House Details,* e.g. real bed,

air bed, couch etc.

Room Type – original categorical variable(attribute) of entity *House Details,* e.g. shared

room, private room.

Property Type – original categorical variable(attribute) of entity *House Details*, e.g.

apartment, hotel, boat, farm stay etc.

Cancellation Policy Type – original categorical variable(attribute) of entity *Policy*, e.g.

strict, flexible, moderate etc.

Amenity – original categorical variable(attribute) of entity *Listing*, e.g. TV, wifi etc.

* Other entities:

House Details – contains the information for listing items in detail.

Review – contains the review record in different aspects for each listing item.

Policy – contains the policy for each listing item to the tenants in different aspects.

Score – contains the score in different aspects provided by the tenants to each listing

item.

**Relationships:**

* Possesses: an association among two entities *Listing* and *House Details.* It represents that house(House refers to listing item for all following contents) possesses its house details. *Listing* and *House Details* both have a key constraint and total participation, i.e. exactly one relationship. Every house (listing) possesses exactly one set of house details, and every set of house details is possessed by exactly one house.
* Possesses\_property: an association among entities *House Details* and *Property Type*. This relationship represents that house details contains the content of property type of house, which is original categorical variable(attribute) of entity *House Details*. *House Details* has a key constraint and total participation, i.e. exactly one relationship, while *Property Type* has total participation, which makes *Property Type* and *House Details* a one-to-many relationship. This means every house detail set must have and only can have one property type information, and every property type information must have at least one house detail set it belongs to, to make it a valid information in this system.
* Possesses\_room: an association among entities *House Details* and *Room Type*. This relationship represents that house details contains the content of room type of house, which is original categorical variable(attribute) of entity *House Details*. *House Details* has a key constraint and total participation, i.e. exactly one relationship, while *Room Type* has total participation, which makes *Room Type* and *House Details* a one-to-many relationship. This means every house detail set must have and only can have one room type information, and every room type information must have at least one house detail set it belongs to, to make it a valid information in this system.
* Possesses\_bed: an association among entities *House Details* and *Bed Type*. This relationship represents that house details contains the content of bed type of house, which is original categorical variable(attribute) of entity *House Details*. *House Details* has a key constraint and total participation, i.e. exactly one relationship, while *Bed Type* has total participation, which makes *Bed Type* and *House Details* a one-to-many relationship. This means every house detail set must have and only can have one bed type information, and every bed type information must have at least one house detail set it belongs to, to make it a valid information in this system.
* Owned by: an association among entities *Listing* and *Host*. This relationship represents that host owns house(s). *Listing* has a key constraint and total participation, i.e. exactly one relationship, while *Host* has total participation, which makes *Host* and *Listing* a one-to-many relationship. This means every house must have and only can have one host, and every host must have at least one house to make them a host in this system.
* Locates\_in: an association among entities *Listing* and *Neighborhood*. It represents that house in the listing locates at certain neighborhood. *Listing* has a key constraint as well as total participation, i.e. exactly one relationship. Every house locates at exactly one neighborhood, which means house must and can only reside in one position, which is obviously true. *Neighborhood* has no constraints such that a neighborhood position can be not located at by any house or be located at by one to many houses. These make *Neighborhood* and *Listing* a one-to-many relationship.
* Locates\_in\_Host: an association among entities *Host* and *Neighborhood*. It represents that house of host locates at certain neighborhood. *Host* has a key constraint as well as total participation, i.e. exactly one relationship. Every house of host locates at exactly one neighborhood, which means house must and can only reside in one position, which is obviously true. *Neighborhood* has no constraints such that a neighborhood position can be not located at by any house or be located at by one to many houses. These make *Neighborhood* and *Host* a one-to-many relationship.
* Has\_a\_neighborhood: an association among entities *Neighborhood* and *City*. This relationship represents that in each city there exits neighborhoods, which are both original categorical variables(attribute) of *Listing* entity. *Neighborhood* has a key constraint and total participation, i.e. exactly one relationship, while *City* has total participation, which makes *City* and *Neighborhood* a one-to-many relationship. This means for every neighborhood, it must exist and can only exist in one city, and every city must have at least one neighborhood to make it a real-life city in this system.
* Has\_a\_city: an association among entities *City* and *Country*. This relationship represents that in each country there exits cities, which are both original categorical variables(attribute) of *Listing* entity. *City* has a key constraint and total participation, i.e. exactly one relationship, while *Country* has total participation, which makes *Country* and *City* a one-to-many relationship. This means for every city, it must exist and can only exist in one country, and every country must have at least one city to make it a real-life country in this system.
* Incurs: an association among entities *Listing* and *Price*. This represents a relationship that listing(house) incurs price in this Airbnb system. *Listing* and *Price* both have a key constraint and total participation, i.e. exactly one relationship. Every house must incur and can only have one price, and a certain price must and can only be incurred by one house.
* Reserves: an association among entities *Listing* and *Calendar*. This represents the relationship that house is reserved on certain date shown on calendar. *Listing* and *Calendar* both have a total participation, while Calendar also has a key constraint, which makes *Listing* and *Calendar* a one-to-many relationship,and each instance in calendar corresponds to exactly one instance in listing. Every house must have information about its availability and price on at least one date on the calendar. Every instance in Calendar must indicate the availability and price for one corresponding Listing instance for the purpose of reservation.
* Receives: an association among entities *Listing* and *Review*. It represents that house receives review from tenants. *Review* has a key constraint as well as total participation, i.e. exactly one relationship, while *Listing* has no constraint. This makes *House* and *Review* a one-to-many relationship. Every house can attain no review, or can attains one to many reviews. Since review is a weak entity of house, if there exists a review, there must have a house and only one house for it to review.
* Has: an association among entities *Listing* and *Policy*. This represents a relationship that house has policy for tenants to obey. *Listing* and *Policy* both have a key constraint and total participation, i.e. exactly one relationship. Every house has exactly one policy for its tenant, and every policy must and only can be owned by one house.
* Has\_cancel: an association among entities *Policy* and *Cancellation Policy Type*. This relationship represents that the Airbnb System policy has a type called cancellation policy, which is original categorical variable(attribute) of *Policy* entity. *Policy* has a key constraint and total participation, i.e. exactly one relationship, while *Cancellation Policy Type* has total participation, which makes *Cancellation Policy Type* and *Policy* a one-to-many relationship. This means every policy must have and only can have one cancellation policy, and every cancellation policy must be contained by at least one policy to make it a part of policies in this system.
* Evaluated: an association among entities *Listing* and *Score*. It represents that house be evaluated by tenants and receives a score. *Listing* and *Score* both have a key constraint and total participation, i.e. exactly one relationship. Every house is evaluated exactly once to achieve a score. Every score must and can only be given to one house after tenants’ evaluation.
* Contains: an association among entities *Listing* and *Amenity*. This relationship represents that house contains amenities inside. There is no constraint for both entities, which makes *Listing* and *Amenity* a many-to-many relationship. Every house can contain no amenity at all, also can have one or many amenities inside. Every kind of amenity can be not contained by any house or can be contained by one to many houses.

## Relational Schema

### ER schema to Relational schema

### Please refer to the description under the corresponding DDL code.

### DDL

## CREATE TABLE Listing

## (

## listing\_id INTEGER(10),

## listing\_url VARCHAR(255) NOT NULL,

## name VARCHAR(255),

## summary VARCHAR(255),

## space VARCHAR(255),

## description VARCHAR(255),

## neighborhood\_overview VARCHAR(255),

## notes VARCHAR(255),

## transit VARCHAR(255),

## access VARCHAR(255),

## interaction VARCHAR(255),

## house\_rules VARCHAR(255),

## pricture\_url VARCHAR(255),

minimum\_nights INTEGER(10),

maximum\_nights INTEGER(10),

host\_id INTEGER(10) NOT NULL,

## neighborhood VARCHAR(255) ,

## city\_id INTEGER(10) NOT NULL,

## country\_id INTEGER(10) NOT NULL,

## latitude DOUBLE(20),

## longtitude DOUBLE(20),

## PRIMARY KEY(listing\_id),

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

FOREIGN KEY(city\_id) REFERENCES Venue(city\_id),

FOREIGN KEY(country\_id) REFERENCES Venue(country\_id),

UNIQUE(listing\_url)

)

The Listing Entity is translated into a table with one primary key, listing\_id, and three foreign keys, host\_id, city\_id and country\_id. With the three foreign keys, we combined the relationship Owned\_by, Located\_in, Located\_at with Listing since each listing will have one unique host, one city and one country. As a result, these columns have NOT NULL constraint on their entry values.

Also, the listing\_url is set to be unique for all instances to ensure proper display of the listing items on the website.

## CREATE TABLE Host

## (

## host\_id INTEGER(10),

## host\_url VARCHAR(255) NOT NULL,

## host\_name VARCHAR(255) NOT NULL,

## host\_since DATE,

## host\_about VARCHAR(255),

## host\_response\_rate FLOAT(10),

## host\_response\_time VARCHAR(255),

## host\_thumbnail\_url VARCHAR(255),

## host\_neighborhood VARCHAR(255),

## host\_verifications VARCHAR(255),

## PRIMARY KEY(host\_id),

## UNIQUE(host\_url),

## )

## CREATE TABLE Neighborhood

## (

neighborhood\_id INTEGER(10),

neighborhood\_name VARCHAR(255),

city\_id INTEGER(10),

## PRIMARY KEY(neighborhood\_id),

FOREIGN KEY (city\_id) REFERENCES City(city\_id)

## )

## CREATE TABLE City

## (

city\_id INTEGER(10),

city\_name VARCHAR(255),

country\_id INTEGER(10),

## PRIMARY KEY(city\_id),

FOREIGN KEY (country\_id) REFERENCES Country(country\_id)

## )

## CREATE TABLE Country

## (

country\_id INTEGER(10),

country\_name VARCHAR(255),

## PRIMARY KEY(country\_id)

## )

## CREATE TABLE Review

## (

## review\_id INTERGER(10),

## listing\_id INTEGER(10) NOT NULL,

## date DATE NOT NULL,

## reviewer\_id INTERGER(10),

## reviewer\_name VARCHAR(255),

## comments VARCHAR(255),

PRIMARY KEY(review\_id, listing\_id),

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

ON DELETE CASCADE

## )

The relationship Receives is combined with the Reviews table as each Reviews corresponds to a unique listing item. As Reviews is designed to be a weak entity of Listing, when one instance of Listing is deleted, the corresponding Reviews instances will be deleted as well.

## CREATE TABLE Score

## (

## score\_id INTERGER(10),

listing\_id INTEGER(10) NOT NULL,

review\_scores\_rating INTEGET(10)

## review\_scores\_accuracy INTEGER(10),

## review\_scores\_clean INTEGER(10),

## reciew\_scores\_checkin INTERGER(10),

## review\_scores\_communication INTERGER(10),

## review\_scores\_location INTERGER(10),

## review\_scores\_value INTERGER(10),

PRIMARY KEY(score\_id),

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

)

The relationship Evaluated is combined with the Score table as each Score corresponds to a unique listing item.

## CREATE TABLE Policy

## (

## policy\_id INTEGER(10),

## is\_business\_travel\_ready BIT,

## cancellation\_policy VARCHAR(255),

## require\_guest\_profile\_picture BIT,

## require\_guest\_phone\_verification BIT,

listing\_id INTEGER(10) NOT NULL,

## PRIMARY KEY(policy\_id),

## FOREIGN KEY (listing\_id) REFERENCES Listing(listing\_id)

## )

The relationship Has is combined with the Policy table as each Policy instance corresponds to a unique Listing instance.

## CREATE TABLE Price

## (

## price\_id INTEGER(10),

## price FLOAT(10),

## weekly\_price FLOAT(10),

## monthly\_price FLOAT(10),

## security\_deposit FLOAT(10),

## cleaning\_fee FLOAT(10),

## guests\_included INTEGER(10),

## extra\_people INTERGER(10),

listing\_id INTEGER(10) NOT NULL,

## PRIMARY KEY(price\_id),

## FOREIGN KEY (listing\_id) REFERENCES Listing(listing\_id)

## )

The relationship Incurs is combined with the Price table as each Price instance corresponds to a unique Listing instance.

## CREATE TABLE House\_Details

## (

## detail\_id INTEGER(10),

## property\_type VARCHAR(255),

## room-type VARCHAR(255),

## accommodates VARCHAR(255),

## bathrooms INTEGER(10),

## bedrooms INTEGER(10),

## beds INTERGER(10),

## bed\_type VARCHAR(255),

## amenities VARCHAR(255),

## square\_feet INTEGER(10),

listing\_id INTEGER(10) NOT NULL,

## PRIMARY KEY(detail\_id),

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

)

The relationship Pocesses is combined with the House\_Details table as each House\_Details instance corresponds to a unique Listing instance.

## CREATE TABLE Amenities

## (

## amenity\_id INTERGER(10),

## amenity\_name varchar(255),

## PRIMARY KEY(amenity\_id)

## )

## CREATE TABLE Contains

## (

## listing\_id INTEGER(10),

## amenitty\_id INTEGER(10),

## PRIMARY KEY(amenity\_id),

## FOREIGN KEY(listing\_id) REFERENCES Listing(listing\_id)

## )

The table Contains is created to store the many-to-many relationship between Amenities and Listing.

CREATE TABLE Calender

{

listing\_id INTEGER(10),

date DATE,

available BIT,

price FLOAT(10),

PRIMARY KEY(listing\_id, date),

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

}

The relationship Reserves is combined with the Calendar table as each Calendar instance corresponds to exactly one Listing instance.

## General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>

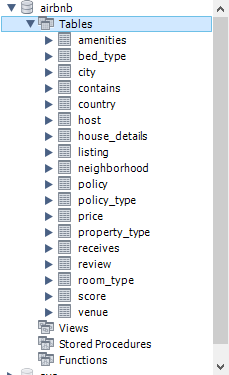
# Deliverable 2

## Assumptions

<In this section write down the assumptions you made about the data. Write a sentence for each assumption you made>

## Data Loading

We have hosted a local database server and created a database named “airbnb”. We’ve created the tables corresponding to the ER models we’ve designed. The tables schemas are coped with the design as well.



We’ve pre-processed the csv data files and organized them with respect to the tables to be created.

To load the data, we adopted the following template of SQL code:

LOAD DATA INFILE

'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/data\_file.csv' (This version requires the loaded file to be only in the secure position designated by the software)

INTO TABLE [*table\_name*]

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

([fields with respect to the columns of the csv files])

(IGNORE LINE 1) (if the first row is the schema)

The data was loaded by executing the commands for all the csv data files.

The screenshot of a portion of the loaded data is as following:



## Query Implementation

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

#### Description of logic:

Select houses that have 8 bedrooms and then obtain the average price of the houses.

#### SQL statement

SELECT AVG(P.price)

FROM House\_Details H, Price P

WHERE H.listing\_id = P.listing\_id AND H.bedrooms = 8

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

#### Description of logic:

Get the amenity id of “TV” and select houses with this amenity id. Then return the average cleaning review score for the houses.

#### SQL statement

SELECT AVG(S.review\_scores\_clean )

FROM Score S, Amenities A, Contains C

WHERE A.amenity\_name = "TV" AND A.amenity\_id = C.amenity\_id AND C.listing\_id = S.listing\_id

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

#### Description of logic:

Find the houses which are available any date between 2019-03-01 to 2019-09-30, and then find the host of the house. Use UNIQUE command to remove duplicate.

#### SQL statement

SELECT UNIQUE H.host\_id, H.host\_name

FROM Host H, Calendar C, Listing L

WHERE H.host\_id = L.host\_id AND L.listing\_id = C.listing\_id AND C.date <=2019-09-30 AND C.date>=2019-03-01 AND C.available = 1

### 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:

Apply self-join to Host table, get the pairs of listing id for hosts who have the same name but different id, that is two different people. Then count the number of listing ids which are found in the union of two pairs set.

#### SQL statement

SELECT COUNT (DISTINCT L.listing\_id)

FROM Listing L

WHERE L.listing\_id IN (

SELECT L1.listing\_id

FROM Host H1, Host H2, Listing L1

WHERE H1.host\_name = H2.host\_name AND

H1.host\_id <> H2.host\_id AND

L1.host\_id = H1.host\_id

UNION

SELECT L2.listing\_id

FROM Host H1, Host H2, Listing L2

WHERE H1.host\_name = H2.host\_name AND

H1.host\_id <> H2.host\_id AND

L2.host\_id = H2.host\_id )

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

#### Description of logic:

Find the houses under Viajes Eco and then find the UNIQUE dates that any of his houses is available.

#### SQL statement

SELECT UNIQUE C.date

FROM Calendar C, Host H, Listing L

WHERE H.host\_name = ‘Viajes Eco’ AND H.host\_id = L.host\_id AND L.listing\_id = C.listing\_id AND C.available = 1

### Query 6: Find all the hosts (host\_ids, host\_names) that have only one listing.

### Description of logic:

Join host and listing tables together with the same host id, and group by host id. Find that for each group of same host id, the number of members, which is the listing owned by the host. The host has only one listing if the number is one.

#### SQL statement

SELECT H.host\_name, H.host\_id

FROM Host H, Listing L

WHERE H.host\_id = L.host\_id

GROUP BY L.host\_id

HAVING COUNT (\*) = 1

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

### Description of logic:

First find the average price of houses with wifi, the the average price of house without wifi. Return the difference of the two values.

#### SQL statement

(

SELECT AVG(P.price)

FROM Price P, Contain C, Amenities A

WHERE A.amenity\_name = "Wifi" AND A.amenity\_id = C.amenity\_id AND C.listing\_id = P.listing\_id

)

- (

SELECT AVG(P.price)

FROM Price P

WHERE NOT EXIST ( SELECT C.listing\_id

FROM Contains C, Amenities A

WHERE A.amenity\_name = "Wifi" AND A.amenity\_id = C.amenity\_id AND C.listing\_id = P.listing\_id)

)

### 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:

First find the average price of house with 8 beds for each city by joining house detail table, listing table, price table and city table. Then use the average price of Berlin to minus that of Madrid.

#### SQL statement

(

SELECT AVG (P1.price)

FROM Price P1, HouseDetail H\_d, Listing L, City C

WHERE C.city\_name =‘Berlin’AND C.city\_id = L.city\_id AND H\_d.beds = 8

AND H\_d.listing\_id = L.listing\_id AND H\_d.listing\_id = P1.listing\_id

)

-(

SELECT AVG (P2.price)

FROM Price P2, HouseDetail H\_d, Listing L, City C

WHERE C.city\_name =‘Madrid’AND C.city\_id = L.city\_id AND H\_d.beds = 8

AND H\_d.listing\_id = L.listing\_id AND H\_d.listing\_id = P2.listing\_id

)

### Query 9: Find the top-10 (in terms of the number of listings) hosts (host\_ids, host\_names) in Spain.

#### Description of logic:

Group the houses by host\_id, and count the number of houses each host that locates in Spain. Rank them according to count in descend order and select the top ten results. Return the corresponding host id and host name.

#### SQL statement

SELECT H.host\_id, H.host\_name

FROM Host H.

WHERE EXISTS (

SELECT TOP 10 H.host\_id, COUNT(

SELECT L.listing\_id

FROM Listing L, Country C

WHERE C.country\_name = "Spain" AND C.country\_id = L.country\_id AND L.host\_id = H.host\_id) as Count

GROUP BY H.host\_id

ORDER BY Count DESC)

### Query 10: Find the top-10 rated (review\_score\_rating) apartments (id,name) in Barcelona.

#### Description of logic:

Group the houses by review score rating, and find the top 10 apartments which are lecated in Barcelona, through the joining of listing, review and city table. Rank them according to count in descend order and select the top ten results. Return the corresponding listing id and listing name.

#### SQL statement

SELECT L.listing\_id, L.listing\_name

FROM Listing L

WHERE EXISTS (

SELECT TOP 10 L.listing\_id, L.listing\_name

FROM Listing L, Review R, City C

WHERE C.city\_name =‘Barcelona’ AND

C.listing\_id = L.listing\_id AND L.listing\_id = R.listing\_id

GROUP BY R.review\_score\_rating

ORDER BY R.review\_score\_rating DESC

)

## Interface

### Design logic Description

<Describe the general logic of your design as well as the technology you decided to use>

### Screenshots

<Provide some initial screen shots of your interface>

## General Comments

<here we briefly explain the changes/improvements we have done from deliverable 1 to deliverable 2>

1. Categorical variables are extracted out from the original entities they belongs to and changed into entities themselves instead of attributes, e.g. Neighborhood, Property\_Type, Room\_Type, Bed\_Type, Cancellation\_policy. As a result, the implementation performance is further optimized since the manipulation of long string is reduced while manipulate integers instead.
2. Relationship between Country, City, Neighborhood together with listing is improved by adding hierarchy(has a relationship) between Country&City, City&Neighborhood. And the new tables created accordingly.
3. ER Model as well as the DDL in deliverable 1 will be further optimized accordingly.

# Deliverable 3

# Assumptions

<In this section write down the assumptions you made about the data. Write a sentence for each assumption you made>

## Query Implementation

<For each query>

### Query a:

#### Description of logic:

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

#### SQL statement

<The SQL statement>

## Query Analysis

### Selected Queries (and why)

#### Query 1

<Initial Running time:

Optimized Running time:

Explain the improvement:

Initial plan

Improved plan>

#### Query 2

<Initial Running time:

Optimized Running time:

Explain the improvement:

Initial plan

Improved plan>

#### Query 3

<Initial Running time:

Optimized Running time:

Explain the improvement:

Initial plan

Improved plan>

# Interface

### Design logic Description

<Describe the general logic of your design as well as the technology you decided to use>

### Screenshots

<Provide some initial screen shots of your interface>

# General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>