Tasks

- (10 pts) Describe the requirements of the problem with a simple document that lists the rules of the database in the problem domain language. Then describe a list of business rules and the list of possible nouns and actions you identified. I'm expecting this to be a short 1- or 2-pages document. This database aim to be an internal tool to manage Airbnb Host.

Business Documentation

Contents

User:

Airbnb Hosts

Airbnb Host can use this database and management application to manage their profile and check their ratings.

Airbnb Staff

Airbnb staff can use this database and application check the status, profile and rating of Hosts. They can also update the rating of Hosts.

Rules:

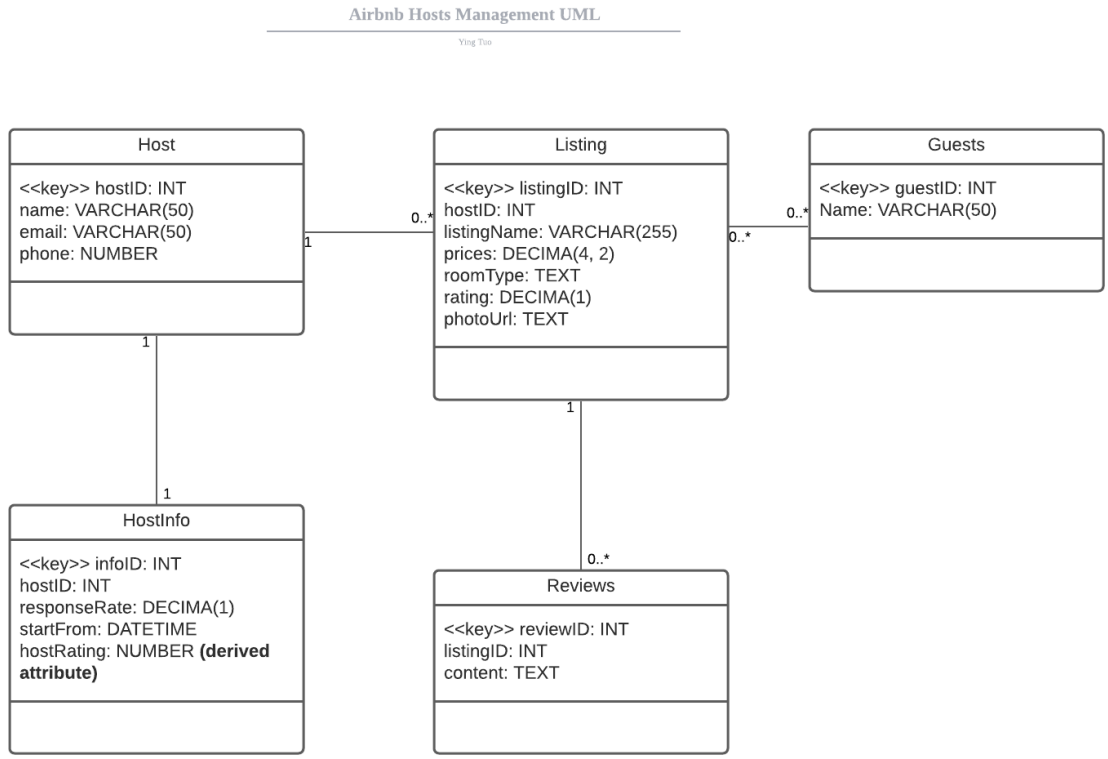
This database aim to be an internal tool to manage Airbnb Host.

- As for the Data schema, I used the data from Airbnb's website: http://insideairbnb.com/get-the-data.html.
- This internal host management tool is used for Airbnb hosts and Airbnb staff to manage the hosts' profile, listing informstion and listing reviews.
- New airbnb hosts can create their account in this application. In the right side of the website, there is a form to submit their name and email to create a new account. Bresides, they will be given a new and unique hostId, which is hidden in this page but will be recorded in the database.
- Airbnb hosts can update their information in this application, for example, update their name and email address

- Airbnb hosts can create new listing in this interface. After they click the "Create New Listing" button, this listing data will be stored in the airbnb.db database.
- Airbnb staff have the access to upddate the rating of hosts based on their listings rating. So, the Rating showed in this page is actually hostRating which is a derived attribute. That means the hostRating does not physical exist in out database, and it is calculated base the the rating attribute in Listings table.
- Both Airbnb staff and Airbnb hosts can submit a review for the lastest listing, and the default rating for this new review is 5.
- The Guests table stores the guests' data of Airbnb.
- Once guests have stayed in one listing, their record will be paired.

Tasks:

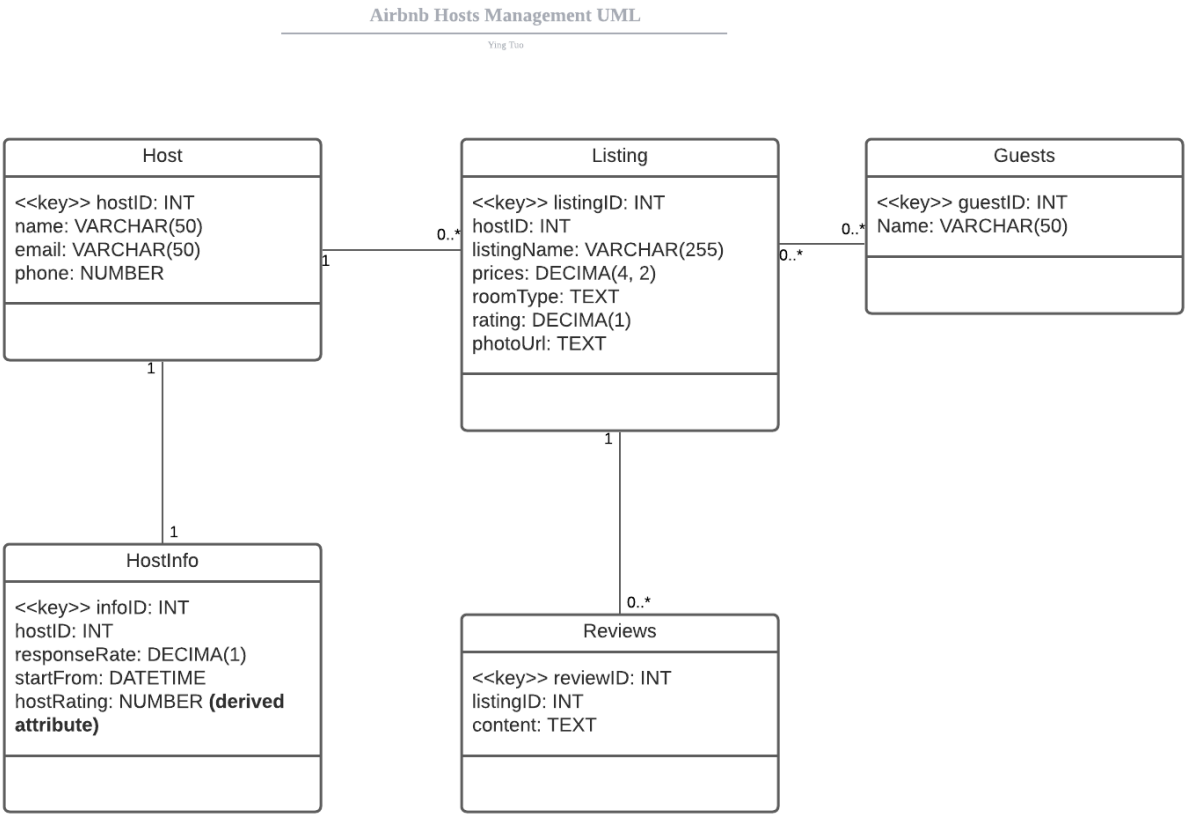
The above rules requires us to create three tables and six routes. I used LucidChart to create UML for these tables,
- The first table is Hosts table.
- The second table is HostInfo table.
- The third table is Listings table.
- The forth table is Reviews table.
- The fifth table is Guests table.



- Our database should build appropriate schema for this database, so that each tables can be associate with others and build relationship.
- In the listings table. **hostID** is the foreign key for this listing. Hence, when we want to delete a host, the corresponding listings with hosted, and reviews in Reviews table with listingid linked with this host should be deleted at the same time. Therefore, we set the foreign keys contains for these foreign keys when create Listings, Reviews schema.
- In order to develop this application, helping Airbnb and Airbnb Host to manage their data, we should have the following APIs to operate our database.
- `/hosts` The home page will redirect to ("/hosts"), which is the main routes. In this page, it shows the Airbnb host database with their name, email, the date they joined Airbnb and most importantly, their hostRating.
- `/hosts/create` The CREATE interface let users to create a new Airbnb user, and post data to database.
- `/hosts/createListing` This route let Airbnb host to create their new listing and post data to database. Besides, because this is a new listing, so I assumen that the default rating for this listing is 5. Later, Airbnb staff can update its rating based on the revires.
- `/hosts/submitReview` This route let Airbnb staff or other users to submit reviews for the newest added lsting and post it to the database.
- `/hosts/delete` This route let us to delete the records in the Hosts table. Besides, because tht hostid in the Hosts table is foreign key in Listings table, and listingid in Listing Table is the foreign key in Reviews Table, all of the related records will bbe deleted.
- `/hosts/update` This route let us to uodate the information and records in the Hosts table.

- (15 pts) Analyze the problem and create a conceptual model in UML using a tool of your choice (e.g., LucidChart, Enterprise Architect, ArgoUML, Visual Paradigm, ERwin, TOAD) as discussed during class and provided in the references and resources below. Additional requirements and clarifications will be provided in the #general channel on Slack. The diagram must contain at least three classes, at least one to many relationship and one many to many. All relationships, except generalization, must have full multiplicity constraints and labeled as appropriate. Classes must have proper names, descriptions, and attributes with domain types. Key attributes and derived attributes must be marked. Don't build a model with more than 10 entities.

I used LucidChart to build my UML. According to this diagram, there are four classes, which are four tables, **Hosts, HostInfo, Listings, Guest,** and **Reviews**. The relationship between Hosts and HostInfo is one to one / zero to one; the relationship between Listings and Reviews is one to many; the relationship between Hosts and Listing is one to one; the relationship between Listings and Guests is many to many.
The description of these attributes is as below, and I build a derived attribute, **hostRating,** which is calculated based on the **rating** attribute in the Listings table. This derived attribute does not physically exist in our database, but it can be derived from other tables and shows in our Web UI.



1. Hosts Table

In this table, we have
<<key>> hostID: INT is our primary key, which is the unique id for Airbnb host;
name: VARCHAR(50) is the name of the Host;
email: VARCHAR(50)is the email of the host.
phone: NUMBER is the phone number of the host.

2. HostInfo Table

In this table, we have
<<key>> infoID: INT is our primary key, which is the unique id for Airbnb host;
hostID: INT is the foreign for this host information;
responseRate: NUMBER is an evaluation criteria for the hosts. Airbnb staff can fill it based on other database;
startFrom: DATETIME denoted the date that the host joined Airbnb
hostRating: NUMBER (derived attribute). It is worthy to notice that this attribute is a derived attribute, which is not actually physically exists in our Airbnb database. This attribute is calculated based on the rating the Listings Table

3. Listing Table

<<key>> listingID: INT is the primary key for this table
hostID: INT is the foreign key for this listing. Noted, when we want to delete a host, the corresponding listings with hostid and reviews with listingid linked with this host should be deleted at the same time. Therefore, we set the contains for this foreign key when create Listings table as below,

FOREIGN KEY("hostid") REFERENCES "Hosts"("hostid") ***ON DELETE CASCADE***

listingName: VARCHAR(255)is the name of this listing
prices: DECIMA(4, 2) is the price/night for this listing
roomType: TEXT can be choose between Private Room and Entire Room;
rating: DECIMA(1)is the attribute that Airbnb staff to fill out based on the reviews of this listing;
photoUrl: TEXT can link to several photo profile of this listing.

4. Reviews Table

<<key>> reviewID: INT is the primary key for this table
listingID: INT is the foreign key for this review. Noted, when we want to delete a host, the corresponding listings with hostid and reviews with listingid linked with this hosted should be deleted at the same time. Therefore, we set the contains for this foreign key when create Reviews table as below,

FOREIGN KEY("listingid") REFERENCES "Listings"(" listingid ") ***ON DELETE CASCADE***

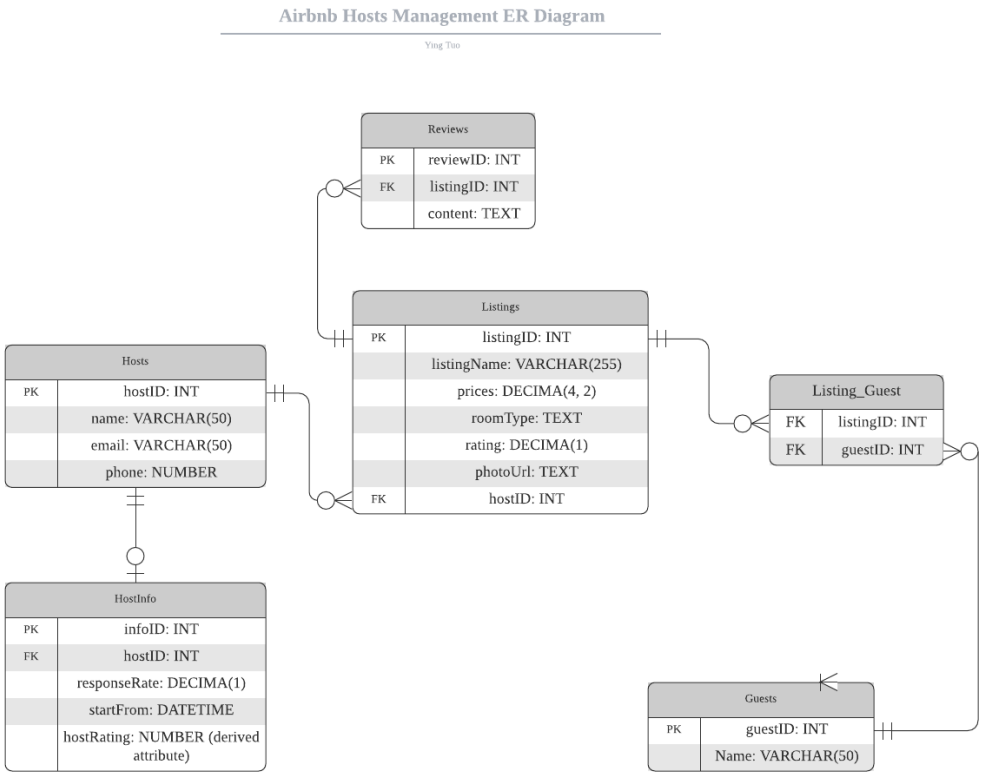
content: TEXT is the content of this review

5. Guests Table

<<key>> guestID: INT is the primary key for this table
name: VARCHAR(50) is the name of guests

6. Listing_Guests Table
listingID: INT holds the one to many relationship with Listings table
guestID: INT holds the one to many relationship with Guests table

- (10 pts) From the Conceptual Model, construct a logical data model expressed as an ERD using a language of your choice (other than UML) and a tool of your choice. The logical data model may not have any many-to-many relationships, so introduce association entities as needed.



As the description in the above question, The relationship between HostInfo to Hosts is **one to one**, while Hosts to HostInfo is **one to zero**; The relationship between Listings to Reviews is **one to many**, while the relationship between Reviews to Listings is **one to one**; The relationship between Hosts to Listings is **zero to many**, while the relationship between Listing to Hosts is **one to many**; the relationship between Listings to Guests is **many to many**.
In order to hold the relationship of many-to-many relationship between Listings to Guests in ERD, we should build an association entities table, Listing_Guest.
Therefore, my ERD will finally have six tables.

- (15 pts) From the logical model, define a relational schema in at least BCNF. Using functional dependencies, show that the schema in in at least BCNF.
The tables in our database are all normal form because they all follow these rules:
“The key” – (1NF) rule: There are no duplicate tuples in tables, which prevent a table from having a primary key. Every table has a primary key.
“The whole key” – (2NF): every attribute must depend on the entire primary key.
“Nothing but the key,” – (3NF): no dependencies on non-key attributes. No value depends on other values that aren't the primary key.
In our database, the hostRating comes from the rating in the Listing table, so it does not need to physically exist in the database, which obeys the 3NF rules.
“No more than one Candidate Key” – (BCNF): for a table to satisfy BCNF, other than table should be in the third normal form, it also should for a dependency A -> B, A should be a super key. Therefore, in my database schema, in the Hosts Table, hostID -> name, email, phone, in words we can say name, email, phone is functionally dependent on hostID. Other tables have the same roles. For HostInfo Table, infoID -> responseRate, startFrom, hostRating; for Listings table, listingID -> hostID, listingName, prices, roomType, rating, photoUrl; for Reviews table, reviewID -> listingID, content; for the Guests table, guestid-> name. Therefore, there is no more than one candidate key

Overall, this logical model has a relational schema in BCNF.

1. Hosts Table

In this table, we have
<<key>> hostID: INT is our primary key, which is the unique id for Airbnb host;
name: VARCHAR(50) is the name of the Host;
email: VARCHAR(50)is the email of the host.
phone: NUMBER is the phone number of the host.

hostID	INT	Primary Key
name	VARCHAR(50)	
email	VARCHAR(50)	
phone	NUMBER	

2. HostInfo Table

In this table, we have
<<key>> infoID: INT is our primary key, which is the unique id for Airbnb host;
hostID: INT is the foreign for this host information;
responseRate: NUMBER is an evaluation criteria for the hosts. Airbnb staff can fill it based on other database;
startFrom: DATETIME denoted the date that the host joined Airbnb

hostRating: NUMBER (derived attribute). It is worthy to notice that this attribute is a derived attribute, which is not actually physically exists in our Airbnb database. This attribute is calculated based on the rating the Listings Table

infoID	INT	Primary Key
hostID	INT	Foreign Key
responseRate	DECIMA (1)	
startFrom	DATETIME	

3. Listing Table

<<key>> listingID: INT is the primary key for this table

hostID: INT is the foreign key for this listing. Noted, when we want to delete a host, the corresponding listings with hostid and reviews with listingid linked with this hosted should be deleted at the same time. Therefore, we set the contains for this foreign key when create Listings table as below,

FOREIGN KEY("hostid") REFERENCES "Hosts"("hostid") ***ON DELETE CASCADE***

listingName: VARCHAR(255)is the name of this listing

prices: DECIMA(4, 2) is the price/night for this listing

roomType: TEXT can be choose between Private Room and Entire Room;

rating: DECIMA(1)is the attribute that Airbnb staff to fill out based on the reviews of this listing;

photoUrl: TEXT can link to several photo profile of this listing.

listingID	INT	Primary Key
hostID	INT	Foreign Key
listingName	VARCHAR (255)	
prices	DECIMA (4,2)	
roomType	TEXT	
rating	DECIMA (1)	
photoUrl	TEXT	

4. Reviews Table

<<key>> reviewID: INT is the primary key for this table

listingID: INT is the foreign key for this review. Noted, when we want to delete a host, the corresponding listings with hostid and reviews with listingid linked with this hosted should be deleted at the same time. Therefore, we set the contains for this foreign key when create Reviews table as below,

FOREIGN KEY("listingid") REFERENCES "Listings"(" listingid ") ***ON DELETE CASCADE***

content: TEXT is the content of this review

reviewID	INT	Primary Key

listingID	INT	Foreign Key
content	TEXT	

5. Guests Table

<<key>> guestID: INT is the primary key for this table
listingID: INT builds the relationship between listings and guests. Since then have many to many relationship, so we need another table, Listing_Guest to build this relationship.
name: VARCHAR(50) is the name of guests

guestID	**INT**	Foreign Key
name	**VARCHAR(50)**	

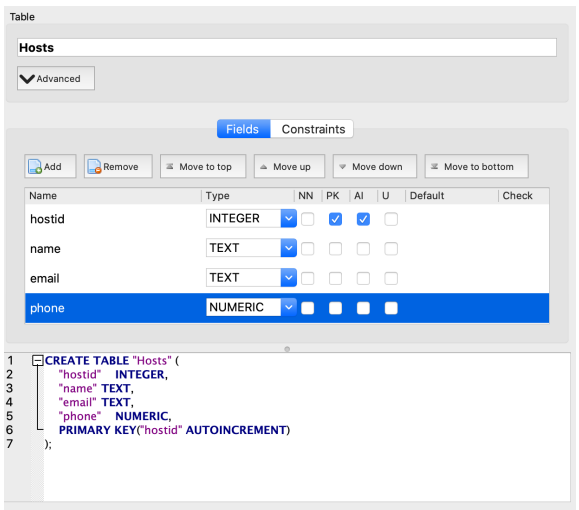
6. Listing_Guests Table

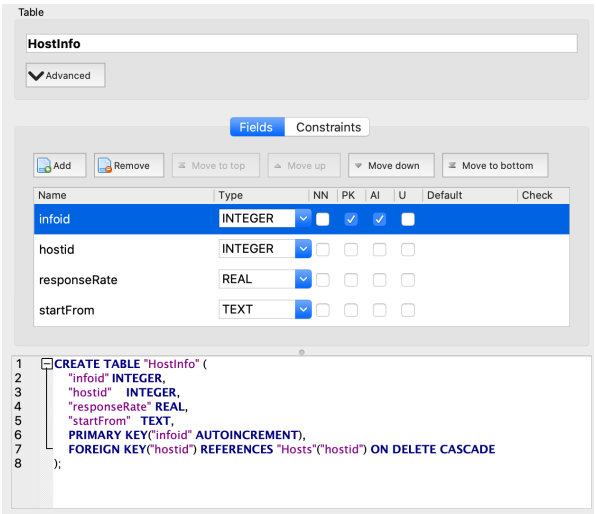
listingID: INT holds the one to many relationship with Listings table
guestID: INT holds the one to many relationship with Guests table

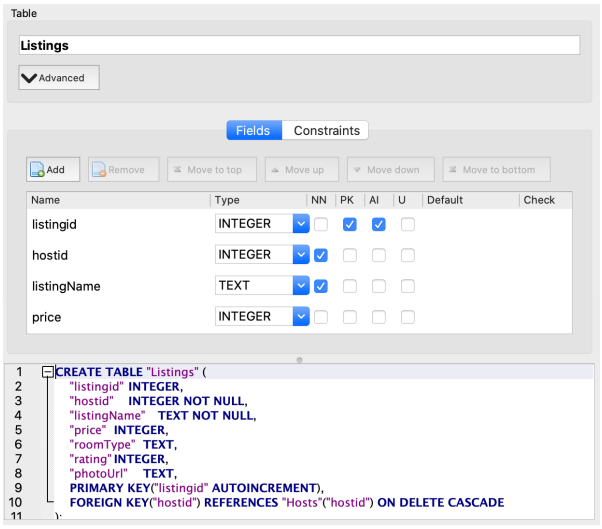
listingID	**INT**	Foreign Key
guestID	**INT**	Foreign Key

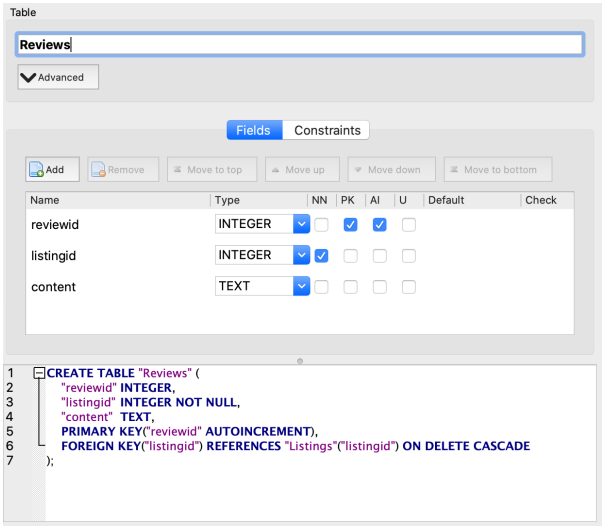
- (10 pts) Create a set of SQL data definition statements for the above model and realize that schema in SQLite3 by executing the script from the SQLite3, the console or Node. You can use DB Browser to generate these statements. Show that the tables were created and conform to the constraints through screen shots or other means.

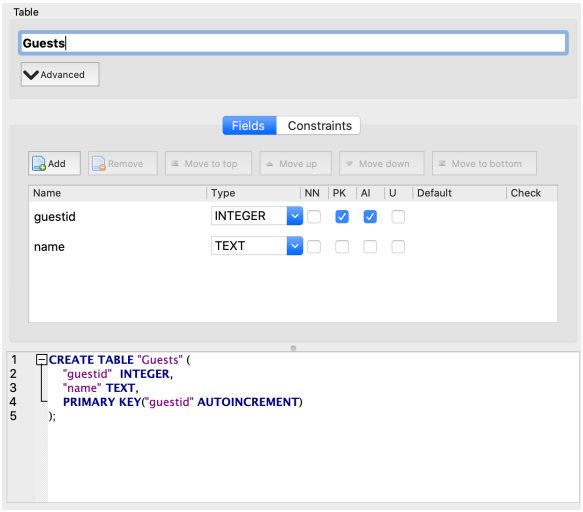
I used the SQLite3 to create these four tables.

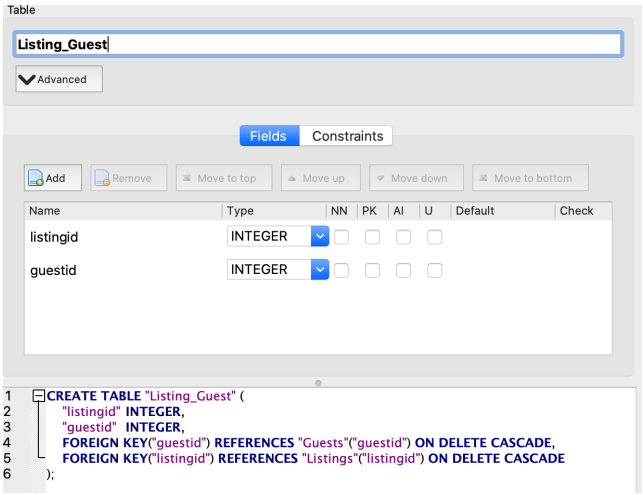






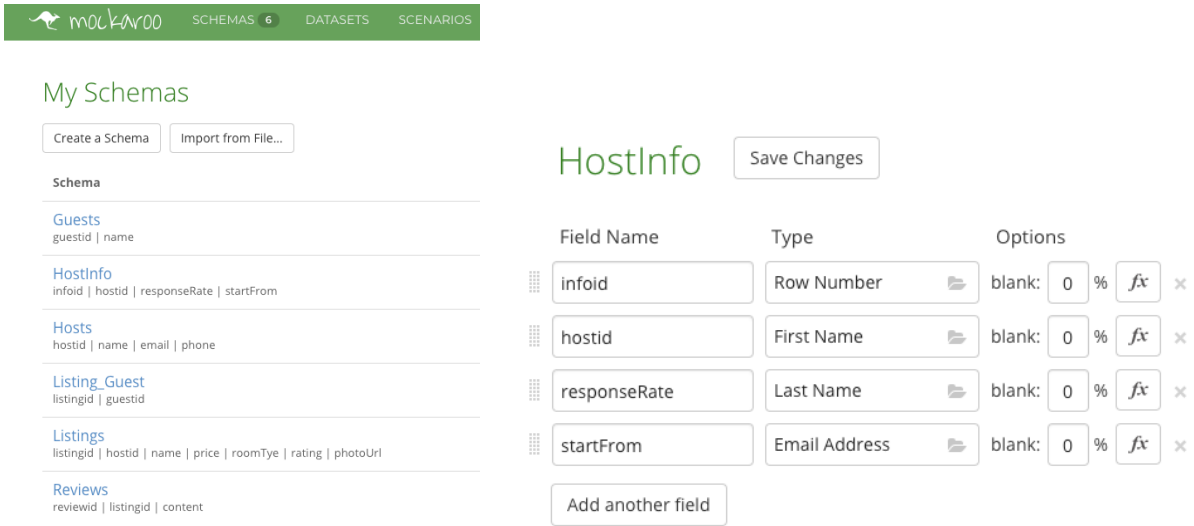


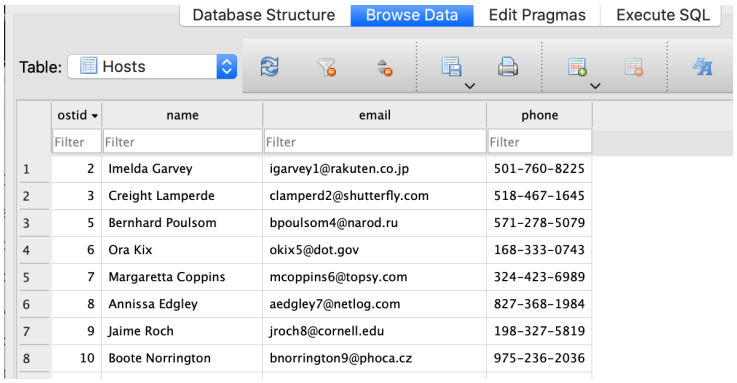




- (10 pts) Populate the tables with test data. You can use tools such as https://www.mockaroo.com/schemas (Links to an external site.)

I used mockaroo to generate test date





	ostid	name	email	phone
	Filter	Filter	Filter	Filter
1	2	Imelda Garvey	igarvey1@rakuten.co.jp	501-760-8225
2	3	Creight Lamperde	clamperd2@shutterfly.com	518-467-1645
3	5	Bernhard Poulsom	bpoulsom4@narod.ru	571-278-5079
4	6	Ora Kix	okix5@dot.gov	168-333-0743
5	7	Margaretta Coppins	mcoppins6@topsy.com	324-423-6989
6	8	Annissa Edgley	aedgley7@netlog.com	827-368-1984
7	9	Jaime Roch	jroch8@cornell.edu	198-327-5819
8	10	Boote Norrington	bnorrington9@phoca.cz	975-236-2036

- (10 pts) Define and execute at least five queries that show your database. At least one query must contain a join of at least three tables, one must contain a subquery, one must be a group by with a having clause, and one must contain a complex search criterion (more than one expression with logical connectors). Experiment with advanced query mechanisms such as RCTE, PARTITION BY, or SELECT CASE/WHEN.

```
-- Queries
-- 1. one query must contain a join of at least three tables,
SELECT Hosts.hostid, Hosts.name, Hosts.email, Hosts.phone, HostInfo.responseRate,
HostInfo.startFrom, Listings.listingName, Listings.price, Listings.rating, Listings.roomType
FROM Hosts, HostInfo, Listings
WHERE Hosts.hostid = HostInfo.hostid AND Hosts.hostid = Listings.hostid;
```

	hostid	name	email	phone	responseRate	startFrom	listingName	price	rating	r
1	2	Imelda Garvey	igarvey1@rakuten.co.jp	501-760-8225	0.7	3/9/18 09:44	Crepis vesicaria L.	94	0	Pr
2	2	Imelda Garvey	igarvey1@rakuten.co.jp	501-760-8225	0.7	3/9/18 09:44	Hieracium scouleri Hook. var. nudicaule (A. Gray) Cronquist	65	0	Pr
3	2	Imelda Garvey	igarvey1@rakuten.co.jp	501-760-8225	0.7	3/9/18 09:44	Iliamna longisepala (Torr.) Wiggins	52	0	En
4	3	Creight Lamperde	clamperd2@shutterfly.com	518-467-1645	0.1	8/27/12 10:23	Crotalaria cunninghamii R. Br.	173	0	Pr
5	3	Creight Lamperde	clamperd2@shutterfly.com	518-467-1645	0.1	8/27/12 10:23	Eryngium pendletonensis K.L. Marsden & M.G. Simpson	154	5	Pr
6	3	Creight Lamperde	clamperd2@shutterfly.com	518-467-1645	0.1	8/27/12 10:23	Helianthella uniflora (Nutt.) Torr. & A. Gray	37	5	En
7	3	Creight Lamperde	clamperd2@shutterfly.com	518-467-1645	0.1	8/27/12 10:23	Senecio amplectens A. Gray var. amplectens	165	5	Pr
8	5	Bernhard Poulsom	bpoulsom4@narod.ru	571-278-5079	0.1	8/26/13 05:16	Eragrostis parviflora (R. Br.) Trin.	94	4	Pr

```
Execution finished without errors.
Result: 496 rows returned in 51ms
At line 10:
SELECT Hosts.hostid, Hosts.name, Hosts.email, Hosts.phone, HostInfo.responseRate, HostInfo.startFrom, Listings.listingName, Listings.price, Listings.rating,
Listings.roomType
FROM Hosts, HostInfo, Listings
WHERE Hosts.hostid = HostInfo.hostid AND Hosts.hostid = Listings.hostid;
```

```sql
-- 2. one must contain a subquery,
SELECT hid, Hosts.name, Hosts.email, ROUND(AVG(listRating), 2) as hostRating
        FROM Hosts,
        (
        SELECT listingid, hostid as hid, Listings.listingName, AVG(Listings.rating) as listRating
        FROM  Listings
        GROUP BY listingid
        )
        WHERE hid = Hosts.hostid
        GROUP BY hid
```

	hid	name	email	hostRating
1	2	Imelda Garvey	igarvey1@rakuten.co.jp	0.0
2	3	Creight Lamperde	clamperd2@shutterfly.com	3.75
3	5	Bernhard Poulsom	bpoulsom4@narod.ru	3.0
4	6	Ora Kix	okix5@dot.gov	4.5
5	7	Margaretta Coppins	mcoppins6@topsy.com	2.0
6	8	Annissa Edgley	aedgley7@netlog.com	2.83
7	9	Jaime Roch	jroch8@cornell.edu	2.5
8	10	Boote Norrington	bnorrington9@phoca.cz	4.0

```
Execution finished without errors.
Result: 174 rows returned in 14ms
At line 8:
SELECT hid, Hosts.name, Hosts.email, ROUND(AVG(listRating), 2) as hostRating
                        FROM Hosts,
                        (
                        SELECT listingid, hostid as hid, Listings.listingName, AVG(Listings.rating) as listRating
                        FROM  Listings
                        GROUP BY listingid
                        )
                        WHERE hid = Hosts.hostid
                        GROUP BY hid
```

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

```sql
-- 3. one must be a group by with a having clause
SELECT hid, name, email, HostInfo.startFrom, hostRating
    FROM HostInfo,
```

```
        (
        SELECT hid, Hosts.name, Hosts.email, ROUND(AVG(listRating), 2) as hostRating
        FROM Hosts,
        (
        SELECT listingid, hostid as hid, Listings.listingName, AVG(Listings.rating) as listRating
        FROM  Listings
        GROUP BY listingid
        )
        WHERE hid = Hosts.hostid
        GROUP BY hid
    )
    WHERE HostInfo.hostid = hid
    GROUP BY hid
    HAVING hostRating >2
```

	hid	name	email	startFrom	hostRating
1	3	Creight Lamperde	clamperd2@shutterfly.com	8/27/12 10:23	3.75
2	5	Bernhard Poulsom	bpoulsom4@narod.ru	8/26/13 05:16	3.0
3	6	Ora Kix	okix5@dot.gov	8/9/18 15:07	4.5
4	8	Annissa Edgley	aedgley7@netlog.com	7/11/12 07:26	2.83
5	9	Jaime Roch	jroch8@cornell.edu	12/15/12 17:29	2.5

```
Execution finished without errors.
Result: 109 rows returned in 21ms
At line 19:
SELECT hid, name, email, HostInfo.startFrom, hostRating
                FROM HostInfo,
                                (
                                SELECT hid, Hosts.name, Hosts.email, ROUND(AVG(listRating), 2) as hostRating
                                FROM Hosts,
                                (
                                SELECT listingid, hostid as hid, Listings.listingName, AVG(Listings.rating) as listRating
                                FROM  Listings
                                GROUP BY listingid
                                )
                                WHERE hid = Hosts.hostid
                                GROUP BY hid
  )
                WHERE HostInfo.hostid = hid
                GROUP BY hid
                HAVING hostRating >2
```

```
-- 4. one must contain a complex search criterion (more than one expression with logical
connectors).
-- Experiment with advanced query mechanisms such as RCTE, PARTITION BY, or SELECT CASE/WHEN.
SELECT *, ROW_NUMBER() OVER(PARTITION BY roomType ORDER BY price DESC) RANK
FROM Listings, Hosts
WHERE Listings.hostid = Hosts.hostid AND price IS NOT NULL
```

	listingid	hostid	listingName	price	roomType	rating	photoUr	hostid	name	email	phone	RANK
232	42	137	Arceuthobium occidentale Engelm.	6	Entire Room	5	http:...	137	Charlotte Boissier	cboissier3s@gizmodo.com	402-929-5745	232
233	245	75	Plagiobothrys scouleri (Hook. & ...	6	Entire Room	0	http:...	75	Becky Stirling	bstirling22@163.com	261-207-1316	233
234	355	159	Pyrola asarifolia Michx. ssp. ...	5	Entire Room	2	http:...	159	Natalina Freebury	nfreebury4e@apache.org	633-325-2625	234
235	346	61	Anacolia menziesii (Turner) Par.	3	Entire Room	2	http:...	61	Mela Goude	mgoude1o@ning.com	347-640-4728	235
236	401	171	Nerisyrenia linearifolia (S. Watso...	3	Entire Room	0	http:...	171	Georgena Edens	gedens4q@google.de	791-655-2060	236
237	257	130	Hymenoxys lapidicola S.L. Welsh...	1	Entire Room	0	http:...	130	Henderson Szachniewicz	hszachniewicz3l@instagra...	619-496-0420	237
238	441	151	Nassella trichotoma (Nees) Hack.	1	Entire Room	0	http:...	151	Krystalle Spatig	kspatig46@oaic.gov.au	251-155-5890	238
239	48	187	Verbascum olympicum Boiss.	200	Private Room	0	http:...	187	Marga Chavrin	mchavrin56@prweb.com	515-934-5002	1
240	107	43	Poa marcida Hitchc.	200	Private Room	4	http:...	43	Brendan Mangenet	bmangenet16@springer.com	992-958-2087	2
241	146	89	Astragalus subcinereus A. Gray	200	Private Room	5	http:...	89	Lev Rapson	lrapson2g@prweb.com	468-689-1833	3
242	354	111	Setaria barbata (Lam.) Kunth	199	Private Room	2	http:...	111	Kris Vreede	kvreede32@nasa.gov	379-608-7273	4
243	365	69	Photinia davidiana (Decne.) Cardot	199	Private Room	1	http:...	69	Elsworth Landrieu	elandrieu1w@fc2.com	278-301-7751	5
244	53	14	Dicranum groenlandicum Brid.	198	Private Room	1	http:...	14	Erinn Halfacree	ehalfacreed@etsy.com	288-334-3806	6
245	179	113	Collema cristatum (L.) F.H. Wigg.	198	Private Room	3	http:...	113	Giff Georgius	ggeorgius34@youtu.be	436-824-3737	7

Execution finished without errors.
Result: 494 rows returned in 24ms
At line 38:
SELECT *, ROW_NUMBER() OVER(PARTITION BY roomType ORDER BY price DESC) RANK
FROM Listings, Hosts
WHERE Listings.hostid = Hosts.hostid AND PRICE IS NOT NULL

```sql
-- 5. Experiment with advanced query mechanisms using SELECT CASE/WHEN.
SELECT
    CASE
        WHEN price >= 100 THEN 'LUXURY'
        WHEN price >= 50 AND price < 100  THEN 'MODERATE'
        WHEN price >= 0 AND price < 50  THEN 'COMFORT'
    ELSE NULL END price_class,
    COUNT(*)
FROM    Listings
WHERE PRICE IS NOT NULL
GROUP BY
    CASE
        WHEN price >= 100 THEN 'LUXURY'
        WHEN price >= 50 AND price < 100  THEN 'MODERATE'
        WHEN price >= 0 AND price < 50  THEN 'COMFORT'
       ELSE NULL END
```

	price_class	COUNT(*)
1	COMFORT	115
2	LUXURY	265
3	MODERATE	114

```
Execution finished without errors.
Result: 3 rows returned in 16ms
At line 43:
SELECT
                CASE
                                WHEN price >= 100 THEN 'LUXURY'
                                WHEN price >= 50 AND price < 100  THEN 'MODERATE'
                                WHEN price >= 0 AND price < 50  THEN 'COMFORT'
                ELSE NULL END price_class,
                COUNT(*)
FROM    Listings
WHERE PRICE IS NOT NULL
GROUP BY
                CASE
                                WHEN price >= 100 THEN 'LUXURY'
                                WHEN price >= 50 AND price < 100  THEN 'MODERATE'
                                WHEN price >= 0 AND price < 50  THEN 'COMFORT'
    ELSE NULL END
```

- (20 pts) Create a basic Node + Express application that let's you create, display, modify and delete at least to of the tables with a foreign key between then. No need to have a polished interface, and you can use the code created in class as a starting point

My repo for this project is: https://github.com/tuoying96/AirbnbHostRate
And my full queries in my project are as below:

```
--Create tables


CREATE TABLE "Hosts" (
    "hostid"    INTEGER,
    "name"  TEXT,
    "email" TEXT,
    "phone" NUMERIC,
    PRIMARY KEY("hostid" AUTOINCREMENT)
);

CREATE TABLE "HostInfo" (
    "hostid"    INTEGER,
    "responseRate"  REAL,
    "startFrom" TEXT
);

CREATE TABLE "Listings" (
    "listingid" INTEGER,
```

```
    "hostid"    INTEGER NOT NULL,
    "listingName"   TEXT NOT NULL,
    "price" INTEGER,
    "roomType"  TEXT,
    "rating"    INTEGER,
    "photoUrl"  TEXT,
    PRIMARY KEY("listingid" AUTOINCREMENT),
    FOREIGN KEY("hostid") REFERENCES "Hosts"("hostid") ON DELETE CASCADE
);


CREATE TABLE "Reviews" (
    "reviewid"  INTEGER,
    "listingid" INTEGER NOT NULL,
    "content"   TEXT,
    PRIMARY KEY("reviewid" AUTOINCREMENT),
    FOREIGN KEY("listingid") REFERENCES "Listings"("listingid") ON DELETE CASCADE
);

CREATE TABLE "Guests" (
    "guestid"   INTEGER,
    "name"  TEXT,
    PRIMARY KEY("guestid" AUTOINCREMENT)
);

CREATE TABLE "Listing_Guest" (
    "listingid" INTEGER,
    "guestid"   INTEGER,
    FOREIGN KEY("listingid") REFERENCES "Listings"("listingid") ON DELETE CASCADE,
    FOREIGN KEY("guestid") REFERENCES "Guests"("guestid") ON DELETE CASCADE
);


-- Queries
-- 1. one query must contain a join of at least three tables,

SELECT Hosts.hostid, Hosts.name, Hosts.email, Hosts.phone, HostInfo.responseRate,
HostInfo.startFrom, Listings.listingName, Listings.price, Listings.rating, Listings.roomType
FROM Hosts, HostInfo, Listings
WHERE Hosts.hostid = HostInfo.hostid AND Hosts.hostid = Listings.hostid;
```

```sql
-- 2. one must contain a subquery,
SELECT hid, Hosts.name, Hosts.email, ROUND(AVG(listRating), 2) as hostRating
        FROM Hosts,
        (
        SELECT listingid, hostid as hid, Listings.listingName, AVG(Listings.rating) as listRating
        FROM  Listings
        GROUP BY listingid
        )
        WHERE hid = Hosts.hostid
        GROUP BY hid

-- 3. one must be a group by with a having clause
SELECT hid, name, email, HostInfo.startFrom, hostRating
    FROM HostInfo,
        (
        SELECT hid, Hosts.name, Hosts.email, ROUND(AVG(listRating), 2) as hostRating
        FROM Hosts,
        (
        SELECT listingid, hostid as hid, Listings.listingName, AVG(Listings.rating) as listRating
        FROM  Listings
        GROUP BY listingid
        )
        WHERE hid = Hosts.hostid
        GROUP BY hid
    )
    WHERE HostInfo.hostid = hid
    GROUP BY hid
    HAVING hostRating >2

-- 4. one must contain a complex search criterion (more than one expression with logical
connectors).
-- Experiment with advanced query mechanisms such as RCTE, PARTITION BY, or SELECT CASE/WHEN.
SELECT *, ROW_NUMBER() OVER(PARTITION BY roomType ORDER BY price DESC) RANK
FROM Listings, Hosts
WHERE Listings.hostid = Hosts.hostid AND price IS NOT NULL

-- 5. Experiment with advanced query mechanisms using SELECT CASE/WHEN.
SELECT
    CASE
        WHEN price >= 100 THEN 'LUXURY'
        WHEN price >= 50 AND price < 100  THEN 'MODERATE'
```

```
        WHEN price >= 0 AND price < 50  THEN 'COMFORT'
    ELSE NULL END price_class,
    COUNT(*)
FROM    Listings
WHERE PRICE IS NOT NULL
GROUP BY
    CASE
        WHEN price >= 100 THEN 'LUXURY'
        WHEN price >= 50 AND price < 100  THEN 'MODERATE'
        WHEN price >= 0 AND price < 50  THEN 'COMFORT'
       ELSE NULL END

-- Read tables
SELECT hid, name, email, HostInfo.startFrom, hostRating
    FROM HostInfo,
        (
        SELECT hid, Hosts.name, Hosts.email, ROUND(AVG(listRating), 2) as hostRating
        FROM Hosts,
        (
        SELECT listingid, hostid as hid, Listings.listingName, AVG(Listings.rating) as listRating
        FROM  Listings
        GROUP BY listingid
        )
        WHERE hid = Hosts.hostid
        GROUP BY hid
    )
    WHERE HostInfo.hostid = hid
    GROUP BY hid
    HAVING hostRating >2

-- Create records
INSERT INTO Hosts(name, email)
    VALUES($Name, $Email);
INSERT INTO HostInfo (hostid, responseRate, startFrom)
    VALUES ((SELECT MAX(hostid) FROM Hosts), $ResponseRate, "10/24/20 17:16");
INSERT INTO Listings (hostid, listingName, rating)
    VALUES ((SELECT MAX(hostid) FROM Hosts), $ListingName, 5);
INSERT INTO Reviews (listingid, content)
    VALUES ((SELECT MAX(listingid) FROM Listings), $Review);

-- Update records
```

```
UPDATE Hosts
    SET
      name = $name,
      email = $email,
      startFrom = $startFrom
    WHERE
      hostid = $hostid;

-- Delete records
DELETE FROM Hosts WHERE hostid==$hostid;
```