CS348 Milestone 2 Report

A brief description of your application on its purpose, use etc

Currently, renting a residence is a headache for people who are living in the Great Toronto Area, especially students and young workers. They either cannot find a room with the desired price, or cannot find a roommate to live together. Also, since there are lots of universities and colleges in GTA, many of the students are having internships or Co-ops. They have to switch residences frequently. We see the demand thus we are trying to provide these people with a platform where they can find not only a rental residence but also potential roommates.

Detailed feature description.

1. When a user enters our website for the first time, the user needs to sign up by creating a user account using their email address, and a unique user id will be generated by our system. The user then can fill up their user account information.
Newly created users will be inserted into the “user” table, where the autogenerated id will be the primary key.
If the user account entered by the user is not registered in our system, a warning message will pop up and the system will ask the user to create a new account.
If the entered password does not match the registered password, a message will pop up and the user will be expected to enter the password again.
SQL: ‘INSERT INTO user (email, `password`) VALUES ('test@test.com', 123456);’
Existing users will be able to login by entering their email address and password.
SQL: ‘SELECT * FROM user WHERE email = ' test@test.com’ and password = 123456";’.

2. The users can interact with the application by applying filters on different features that they would like to be in their residence. For example, users can choose their target region, desired number of bedrooms, bathrooms and den. Users can also search by the price range of the residence.

The result of housing information will be sorted by the popularity of the house, i.e., how many people added this house to their wish list. A new field `pop` will be added to the `house` table to keep track of the popularity. We will implement this new change by adding a trigger to the table `wish_list`, i.e., update `pop` in `house` whenever a user adds a house to wish list. `pop` will be 0 at the initial state.
SQL: SELECT * FROM houses WHERE address LIKE '%Markham%'
AND bathroom = 1 AND bedroom = 1 AND den = 0 AND price < 2500
AND price > 1500 AND pet_friendly = 1;
SQL: CREATE TRIGGER addPop AFTER INSERT ON wish_list
REFERENCING NEW ROW AS r FOR EACH ROW UPDATE house AS h
SET pop = pop + 1 WHERE h.id = r.hid

We are interested in implementing more searching criteria like parking, pet-friendly and so on in the future.

3. Users who want to rent their house can choose to post their own rental information. Users who only want to search for a preferred housing are also supported.

SQL: INSERT INTO houses (bedroom, bathroom, den, address, lat, `long`, price) VALUES(2 , 2 , 1, 'M4L1G3, Toronto',43.66993150,-79.37546270, 3000);

4. Users will be able to add houses to their wish list when they find desired houses. A new table 'wish_list' will be used for this. The table has two fields, 'hid' and 'uid'.
Users can leave a message on the house they added to their wish list so that potential roommates can see those messages.
SQL: INSERT INTO wish_list (uid, hid,message) VALUES(1,123,"hello world")

5. Each house will have an additional information of users who added this house to their wish list and message they left so that they could potentially rent together.
SQL: SELECT wl.uid, u.email, wl.message FROM wish_list wl LEFT JOIN user u ON u.id = wl.uid WHERE uid != 1 AND hid = 3;

6. A rating system will be included in the application. Users will be able to rate a residence (from 1 to 5) and leave a comment if they have lived there before

(we cannot check this though). The rating information will be included in the housing information so that everyone can see the rating(we will take the average of all ratings) and make a better decision on whether they want to rent this house. As a result, a new table `rating` will be added.
SQL: INSERT INTO `rating` (uid, hid, rate, comment) VALUES(1,123,2,"hello world")

7. A user can find the owner of the house in their wish list. A user can contact with the owner through the owner's user email.
SQL: SELECT u.email FROM wish_list AS w, houses AS h, user AS u WHERE h.id = w.hid AND u.id = h.owner;

Detailed plan for getting data

We intend to get our data from Kaggle. The primary data we are using is retrieved from https://www.kaggle.com/rajacsp/toronto-apartment-price, which contains housing information in GTA region. The housing information contains the number of bedrooms, bathrooms and dens, address, latitude, longitude, price. Note that we might obtain more data or add more fields into our dataset in the future.
In order to implement all the features mentioned above, we added three attributes to the original dataset, namely "Pet_friendly", "Parking" and "pop". The data for the first two attributes are randomly generated, the data for "pop" is initialized to be 0.
The sample data is as follows. (Note: 0 indicates NO and 1 indicates YES for Pet_friendly and Parking.)

id	Bedroom	Bathroom	Den	Address	Lat	Long	Price	Pet_friendly	Parking	pop	owner
1	**2**	2	0	3985 Grand Park Drive, 3985 Grand Park Dr, Mississauga, ON L5B 0H8, Canada	43.5816391	-79.648193	$2,450.00	0	1	0	1

2	**1**	1	1	361 Front St W, Toronto, ON M5V 3R5, Canada	43.6430505	-79.3916429	$2,150.00	1	1	0	2
3	**1**	1	0	89 McGill Street, Toronto, ON, M5B 0B1	43.6606054	-79.3786354	$1,950.00	0	0	0	3

The dataset is in CSV format and there are two ways of importing the data

1. Using SQL query LOAD DATA LOCAL INFILE . Be more specific:

```
LOAD DATA LOCAL INFILE 'path/to/file/houses.csv'
INTO TABLE table_name
FIELDS TERMINATED BY ','
OPTIONALLY ENCLOSED BY '"'
LINES TERMINATED BY '\n'
IGNORE 1 ROWS;
```

2. Using external Database tool like MySQLWorkbench or DBeaver and directly import the csv file through the interface.

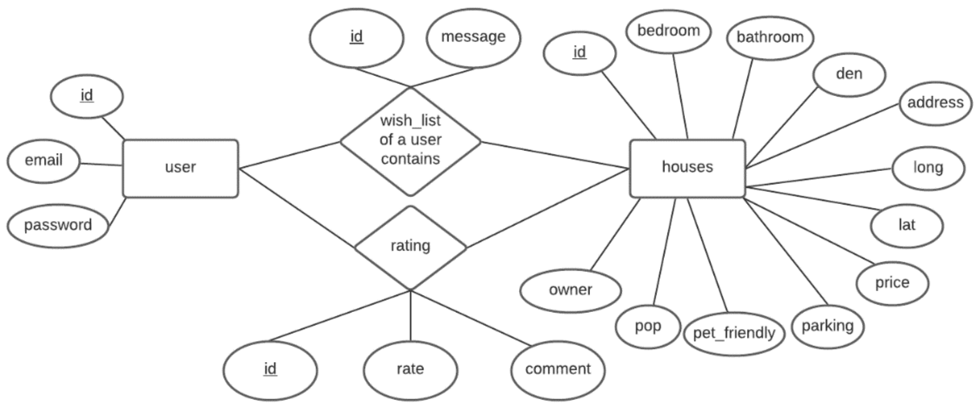
Detailed description of system support

PHP will be the primary language we use for our project to handle the backend data and MySQL is the backend service we use. We did not decide to employ any pre-existing frontend framework yet, but we will probably use it in the future. Also, some javascript will be employed to handle frontend tasks

A list of assumptions

○ User id is unique in the 'user' table, and house id is unique in the 'houses' table.
○ There is no missing data in original imported data.
○ 'bedroom', 'bathroom' and 'den' values in 'houses' table are non-negative integers.
○ Each 'uid' and 'hid' in 'wish_list' table must appear in 'user' and 'houses' tables, respectively.
○ The owner in houses must be an id in user table.

An E/R diagram for your database design.



A list of database tables with keys declared (relational data model).

user(<u>id</u>, email, password);
houses(<u>id</u>, bedroom, bathroom, den, address, lat, long, price, pet_friendly, parking, pop, owner);
wish_list(<u>id</u>, uid, hid, message);
rating(<u>id,</u> uid, hid, rate, comment);
The uid in wish_list and rating relation must be contained in the user relation.
The hid in wish_list and rating relation must be contained in the houses relation.
The owner in houses must be an id in user table.

Changes you made to the database during performance tuning in Task 8, e.g., additional indexes created

We plan to use ISAM to manage a large size of data.
In relation ‘wish_list’ and ‘rating’, we choose id to be the clustering index. For non-clustering index, we wish to create an additional secondary index on hid for both relation:
CREATE INDEX wishlistHouseIndex ON wish_list(hid);
CREATE INDEX ratingHouseIndex ON rating(hid);

Therefore, we use sparse index on id, and dense index on hid for ‘wish_list’ and ‘rating’ to handle large dataset and potential performance issues.
For other relations, we decide to use primary key as the clustering index, all records are clustered by the primary key. Additional secondary indices are not decided yet, we might change it during the implementation process.

SECURITY:
To ensure that there will not be leakage from our database, we add extra security measures to prevent html and sql injection. Instead of reading the input directly, we use statements like:
$bedroom = (int)$_POST['bdrm'];
select * from houses where bedroom = '$bedroom';
to disallow inputting html and sql clauses in the search form. This helps protecting private data from our users.

Members:

Chenxingyu Su: Improve database schemas, design new features
Mason Ma: Modify SQL queries, design new features
Bingjie Shen: Update the dataset according to our requirement and new features, design new features
Jiayi Wang: Update E/R diagram, design new features