#placeholder for first page

2) Project Responsibilities:

Joel Tan Wan Rong: Contributed to the design of the ER data model, coded the DDL statements, the non-trivial constraints, and the complex queries. Writing the parts of the report related to the database.

3) Application Requirements and Functionalities

- When a user registers, they will specify whether they want to be a Pet Owner or a Caretaker. Every user must be exactly one of the 2 user types. (Covering constraint + No overlap constraint).

- Pet owners can create profiles for their pets, and each pet profile contains various information, notably their species (i.e. Dog, Cat, etc.) and their weight class. Weight class is determined as follows:

‘small’ (<10kg)

‘medium’ (10kg-25.9kg)

‘large’ (26kg-44kg)

‘giant’ (>44kg)

The species type and weight class information have default values of ‘unknown’ if the owner does not know or does not want others to know this information about their pet.

- Each caretaker can offer multiple care types, with each care type specifying a unique combination of the pet species, the weight class of the pet and the service type. Service type includes:

‘hotel’: pet stays at the caretaker’s location

‘petsitting’: pet and caretaker stays at the owner’s location

‘drop-in visits’: the caretaker checks in on the pet at the owner’s location once in awhile

‘daycare’: day care at the caretaker’s location for when the owner is at work

The caretaker can add additional descriptions to the care type.

- Owners and Caretakers can chat among each other. (Owners cannot chat with other owners, and caretakers cannot chat with other caretakers.)

- Caretakers can advertise their availabilities in the form of time intervals in days. (i.e. from 21/3/19 to 3/4/19). Overlapping (with respect to days) availabilities by a particular caretaker will be merged in the database.

- Owners can filter for availabilities that suit their needs, and can place bids for those availabilities. The owner specifies a time interval for the days that he needs the caretaker’s services, as long as the owner’s start and end dates fall within the availabilities’ time interval. The owner can give additional descriptions for each bid as well. (such as how many pets, what service he is bidding for, etc.)

- Caretakers can accept as many bids as possible. Each accepted bid is one successful caretaking service. Owners can rate the caretaker’s services and the caretaker can rate the owner’s conduct. They can also leave each other comments if they want to.

- Each payment by the owner is linked to the accepted bid which the owner is paying for. The owner may pay in multiple transactions.

#add more features

4) ER model diagram

#placeholder for diagram

Constraints not present in the ER model:

1. We prevent having overlapping availabilities by the same caretaker.
2. For the Bids table, we need ostart\_date >= referenced availability's start\_date and oend\_date <= referenced availability's end\_date.

5) Relational schema

#placeholder for a lot of code

6) There were more than 3 triggers but we will highlight the three most relevant.

Trigger 1: An owner cannot also be a Caretaker. Similarly, a caretaker cannot also be an owner (code not shown but almost identical to below). This is to preserve the no-overlapping constraint.

**CREATE** **OR** **REPLACE** **FUNCTION** not\_owner()

**RETURNS** **TRIGGER** **AS**

**$$**

**BEGIN**

**IF** **EXISTS** (

**SELECT** 1

**FROM** Owners

**WHERE** Owners.username = **NEW**.username

) **THEN** **RETURN** **NULL**;

**ELSE** **RETURN** **NEW**;

**END** **IF**;

**END**;

**$$** **LANGUAGE** plpgsql;

**CREATE** **TRIGGER** not\_owner\_trig

**BEFORE** **INSERT** **OR** **UPDATE**

**ON** Caretakers

**FOR** **EACH** **ROW**

**EXECUTE** **PROCEDURE** not\_owner();

Trigger 2: For each bid to a particular availability, the owner must input a start and end date that is between the start and end dates of the availability which was advertised by the caretaker. This is to prevent non-sensical bids where the owner bids for a time period which is not within the availability’s time period.

**CREATE** **OR** **REPLACE** **FUNCTION** valid\_bid\_date()

**RETURNS** **TRIGGER** **AS**

**$$**

**BEGIN**

**IF** (

**NEW**.ostart\_date < (**SELECT** start\_date **FROM** Availabilities A **WHERE** A.id = **new**.availabilityId) **OR**

**NEW**.oend\_date > (**SELECT** end\_date **FROM** Availabilities A **WHERE** A.id = **new**.availabilityId)

) **THEN** **RETURN** **NULL**;

**ELSE** **RETURN** **NEW**;

**END** **IF**;

**END**;

**$$** **LANGUAGE** plpgsql;

**CREATE** **TRIGGER** valid\_bid\_date\_trig

**BEFORE** **INSERT** **OR** **UPDATE**

**ON** Bids

**FOR** **EACH** **ROW**

**EXECUTE** **PROCEDURE** valid\_bid\_date();

Trigger 3: A caretaker should not have any availabilities with overlapping time periods. The trigger below handles the logic to merge any overlaps at the time of insertion or update. Note that it is possible for the new availability to overlap with multiple other availabilities. Also note that the bids which refer to the original availabilities (which had been part of the overlap) will be updated to refer to the new, merged availability.

**CREATE** **OR** **REPLACE** **FUNCTION** merge\_availability()

**RETURNS** **TRIGGER** **AS**

**$$**

**DECLARE** datecursor **CURSOR** (new\_start\_date **DATE**, new\_end\_date **DATE**) **FOR**

**SELECT** \*

**FROM** Availabilities A

**WHERE** A.start\_date <= new\_end\_date **AND** A.end\_date >= new\_start\_date **AND** A.ctname = **new**.ctname;

final\_start\_date **DATE**;

final\_end\_date **DATE**;

availability RECORD;

**BEGIN**

**OPEN** datecursor(new\_start\_date := **new**.start\_date, new\_end\_date := **new**.end\_date);

final\_start\_date := **new**.start\_date;

final\_end\_date := **new**.end\_date;

--first loop to extract the final start\_date and end\_date.

**LOOP**

**FETCH** datecursor **INTO** availability;

EXIT **WHEN** **NOT** **FOUND**;

final\_start\_date := **LEAST**(availability.start\_date, final\_start\_date);

final\_end\_date := **GREATEST**(availability.end\_date, final\_end\_date);

**raise** **notice** 'a';

**END** **LOOP**;

**MOVE** **BACKWARD** **ALL** **FROM** datecursor;

**ALTER** **TABLE** Bids **DISABLE** **TRIGGER** **ALL**; --temporarily disable constraindate for Bids so that id can be altered.

--second loop to update Bids (which has a foreign reference to Availabilities) and delete entries in Availabilities which are in the overlap.

**LOOP**

**FETCH** datecursor **INTO** availability;

EXIT **WHEN** **NOT** **FOUND**;

**UPDATE** Bids B

**SET** availabilityId = **new**.id

**where** B.availabilityId = availability.id;

**DELETE** **FROM** Availabilities A **WHERE** **CURRENT** **OF** datecursor;

**raise** **notice** 'b';

**END** **LOOP**;

**CLOSE** datecursor;

**ALTER** **TABLE** Bids **ENABLE** **TRIGGER** **ALL**;

--finally insert 1 entry into Availability which encompasses all the deleted entries as well as the newest entry.

**RETURN** (**new**.id, **new**.ctname, final\_start\_date, final\_end\_date);

**END**;

**$$** **LANGUAGE** plpgsql;

**CREATE** **TRIGGER** merge\_trig

**BEFORE** **INSERT** **OR** **UPDATE** **ON** Availabilities

**FOR** **EACH** **ROW**

**EXECUTE** **PROCEDURE** merge\_availability();

7) Description of our three interesting queries:

Query 1:

This query provides insight on the following useful analytical question with respect to the caretakers: How does the caretaker's rating affect the bidding prices of bids on his availabilities?

The query finds the average bidded price per hour for each caretaker rating group, where the caretakers are grouped by their average rating (rounded to 0.1).

If there is no caretaker with a particular rating group, average bidded price per hour value for that group is NULL.

**with** CTRatingsRounded **as** (

**select** username **as** ctname, **round**(rating, 1) **as** rounded\_rating

**from** UserRating

**where** is\_owner = **false**

),

RatingGroups **as** (

**select** **generate\_series**(1,5,0.1) **as** rounded\_rating

)

**select** rounded\_rating **as** rating, **avg**(bidded\_price\_per\_hour) **as** avg\_bidded\_price\_per\_hour

**from** RatingGroups **natural** **left** **join** ((Bids B **join** Availabilities A **on** B.availabilityId = A.id) **natural** **join** CTRatingsRounded)

**group** **by** rounded\_rating

**order** **by** rounded\_rating

;

Query 2:

This query provides insight on another question, this time with regards to the pet owners, namely: How does the owner's rating affect the rate at which his bids are accepted?

The following query finds the proportion of bids accepted for each owner rating group, where the owners are grouped by their average rating (rounded to 0.1).

If there are no bids from owners with a particular rating group, proportion of accepted bids for that group is NULL.

**with** ORatingsRounded **as** (

**select** username **as** oname, **round**(rating, 1) **as** rounded\_rating

**from** UserRating

**where** is\_owner = **true**

),

RatingGroups **as** (

**select** **generate\_series**(1,5,0.1) **as** rounded\_rating

)

**select** rounded\_rating **as** rating, **case**

**when** **count**(B.id)>0 **then** **cast**(**count**(AB.id) **as** **numeric**)/**count**(B.id)

**else** **null**

**end** **as** prop\_accepted\_bids

**from** RatingGroups **natural** **left** **join** ((Bids B **left** **join** AcceptedBids AB **on** B.id = AB.id) **natural** **join** ORatingsRounded)

**group** **by** rounded\_rating

**order** **by** rounded\_rating

;

Query 3:

This last query asks the following: How does the frequency of a caretaker's availability affect his popularity?

If the frequency of a caretaker's availability is independent to his popularity, one might expect that after adjusting for caretaker ratings and other potential confounders, for any given caretaker, the average number of bids placed per day of his availability, would be roughly constant.

It would be a sensible claim to argue that the above independence is false, as one would expect a caretaker who is frequently available to garner more popularity over time. We seek to analyse if this is in fact the case.

The following query finds for a particular caretaker, the value of the average number of bids placed per day of his availability.

**with** AvailabilitiesWithNumBids **as** (

**select** ctname, (end\_date - start\_date + 1) **as** interval\_length\_in\_days, (

**select** **count**(\*)

**from** Bids

**where** availabilityId = A.id

) **as** num\_bids

**from** Availabilities A

)

**select** username, **case**

**when** **count**(interval\_length\_in\_days) > 0 **then** **sum**(num\_bids)/**sum**(interval\_length\_in\_days)

**else** **NULL**

**end** **as** num\_bids\_per\_day

**from** Caretakers CT **left** **join** AvailabilitiesWithNumBids A **on** CT.username = A.ctname

**group** **by** username

;

8) Specification of software tools/frameworks used in the project:

#