NUS | Computing

National University of Singapore

Group 25's CS2102 Report

Pet Society

Ong Bik Jeun A0206349A
Ooi Jun Hao A0199461R
Gu YunTian A0199586B
Wang Ziyue A0201879X
Brendan Wan Shi Jie

Project Responsibilities

Name	Responsibilities
Ong Bik Jeun	1) Creation of ER diagram 2) SQL Create tables 3) SQL Query statements 4) SQL trigger statements 5) Report Write up -> Section 1/2/4/5/6
Ooi Jun Hao	1) Creation of ER diagram 2) Web functionalities development 3) SQL queries & triggers
Gu YunTian	1) Creation of ER diagram 2) Draft web page outline 3) Draft web program flow 4) Report write up -> Section 3 5) Project Video Demo Creation
Wang Ziyue	1) Report write up -> Section 7 2) Project Video Demo Creation
Brendan Wan Shi Jie	

1 Introduction

Our project is to create a Pet Caring Service which connects to a SQL database. This system allows pet owners to search for care takers for their pets for a certain period of time. Users which are classified as admin, caretaker and pet owners have their own accounts and have access to multiple functionalities they can use respectively to achieve this purpose. Users can advertise their availability (when they are available to work, how much are they proposing, etc) or can surf through the possible candidates and bid for the other users which they wish to hire to fulfil their service.

1.1 Implementation Specification

In our project, we developed the backend using PostgreSQL and NodeJS while our frontend is developed using bootstrap. JavaScript is our main language for this project. We adopted a MVC architectural pattern. The model is stored in a database where it is used to update the view and can be updated using the controller. The view handles what the user sees and is updated by the model. This will be our .ejs files. Finally the controller handles user interaction, where users are able to use this interface to manipulate the models.

2 Database Design

2.1 Data Requirements and Functionalities

Our system ensures that each user has an account and is logged in before they have access to the respective functionalities. Hence, our system supports the creation/deletion/updating for the different users. It also supports the data access for the different users such as viewing reviews for care takers by pet owners, the past jobs and ratings. In turn the caretakers can also view their own history log and their own personal details like their salary. To allow for an easy process of the bidding, pet owners will be able to browse and search for caretakers and place a bid for them. Prior to this, the caretakers would first be able to advertise their availability up to the system which will be displayed for the pet owners to see. An important function that is implemented is allowing the caretakers to have more than 1 job service at a given time, however this is also further separated based on part timer and full timer, where part timers job offers are determined by their ratings. Hence all these will have to be calculated and tracked and stored in our database. Another major functionality we have implemented in our project would be the calculation of the base daily price which in turn will affect the salaries of the caretakers. Our data must support the updating of the price and salary count for the individual caretakers in order to account for the accuracy of the data processing.

In general, our system also supports the viewing of the summary information relevant to each user. For instance, caretakers are able to view the summary of the total number of pet days or calculate for the expected salary. Pet Owners are able to browse for all caretakers around their area or browse their own pet information. Whereas the PCS Admin will be able to browse the summary information of the total number of pets taken care in the month, total salary to be paid to caretakers, etc.

2.2 Data Constraints

1) Users are uniquely identifiable by their email. Address, name, zone and password are recorded as well.
2) Users can be a PetOwner, a CareTaker or a PCSAdmin. A user can both be a PetOwner as well as a CareTaker at once.
3) Each PetOwner can own more than 1 Pet.
4) Each PetOwner may own a Pet but each pet must be owned by 1 pet owner.

5) Pet is an identity dependent weak entity set with Pet Owner as the owning entity. Name, special requirements, pet type and base daily price must be recorded.

6) Each pet will be uniquely identifiable by owner email, pet name and the pet type.

7) Each PetOwner can possess more than 1 Credit Card.

8) Each Credit Card will be owned by a pet owner

9) Credit Card is an identity dependent weak entity set with Pet Owner as the owning entity. The card number, expiry date and cvv is recorded.

10) Each Credit Card is uniquely identifiable by the owner email and the card number.

11) Each Caretaker is either a PartTimer or FullTimer.

12) Part Timers must have a whole set of availabilities recorded for the given year

13) Each availability is an identity dependent weak entity set with Part Timer as the owning entity. The date will thus be recorded

14) Each availability will be associated with 1 part timer

15) Each availability will be uniquely identifiable by the caretaker email and the date.

16) Fulltimers can apply for multiple leaves.

17) Each leave will be associated with 1 fulltimer

18) Leaves is an identity dependent weak entity set with Full timer as the owning entity. Start and end date and the reason for each leave must be recorded

19) Pet Owners will place a bid for caretakers for a service.

20) Each Service provided will have their pickup date, duration and price recorded.

21) Each service will only be between 1 pet owner and 1 caretaker with a specific bid agreement.

22) Each bid agreement will have its transport_agreement, rating, review and accepted status stored.

23) Each rating and review for each specific transaction can only be done after the service end date

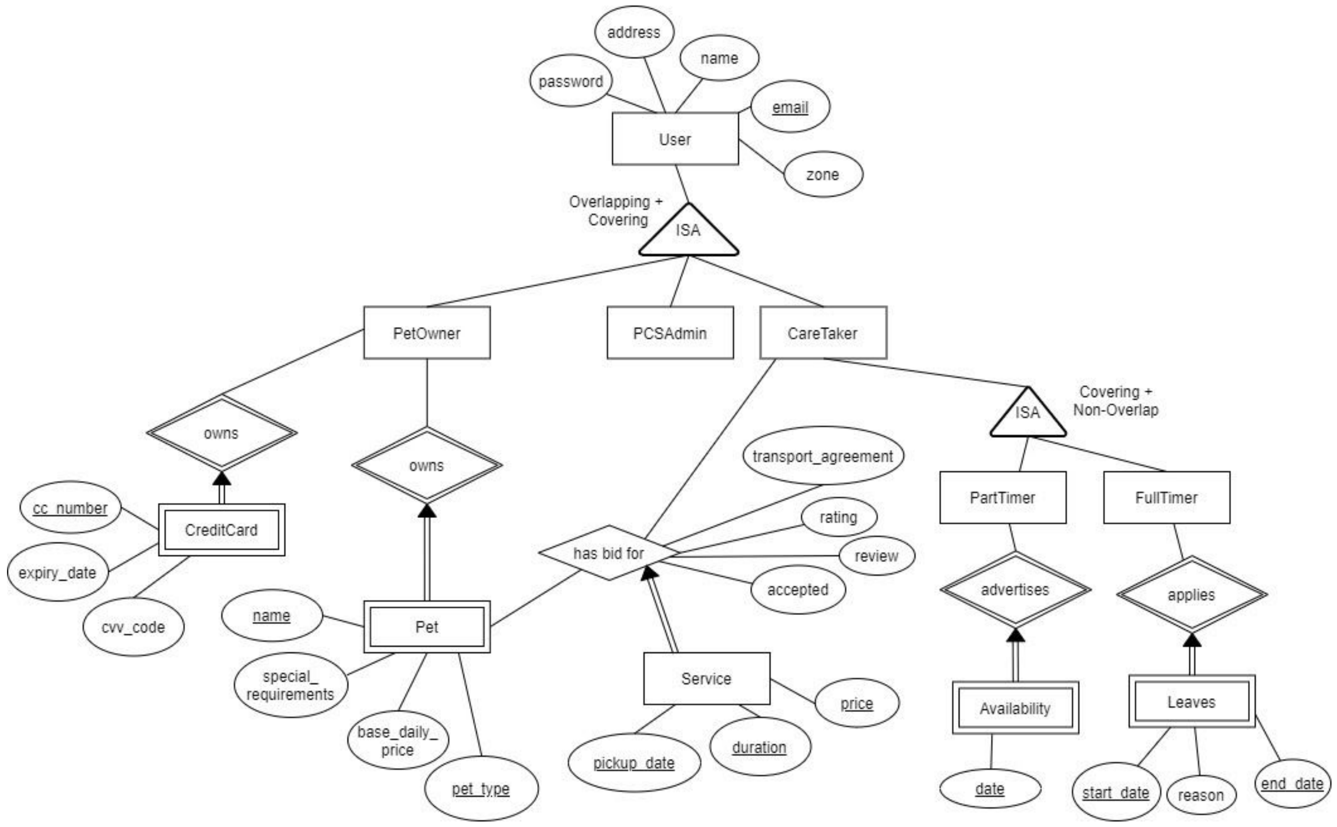
24) Each full timer must have a minimum of 2 x 150 consecutive days of work a year

25) A leave record cannot be generated if the dates overlap with an accepted service

26) There can only be a maximum of 5 ongoing service for full time caretakers and part timers if their average rating is >= 4. Else, only a maximum of 2 is allowed.

27) Full timer’s bid will be automatically accepted whenever possible

2.3 ER Model



Data Constraints not captured in ER model:

23) Each rating and review for each specific transaction can only be done after the service end date

24) Each full timer must have a minimum of 2 x 150 consecutive days of work a year

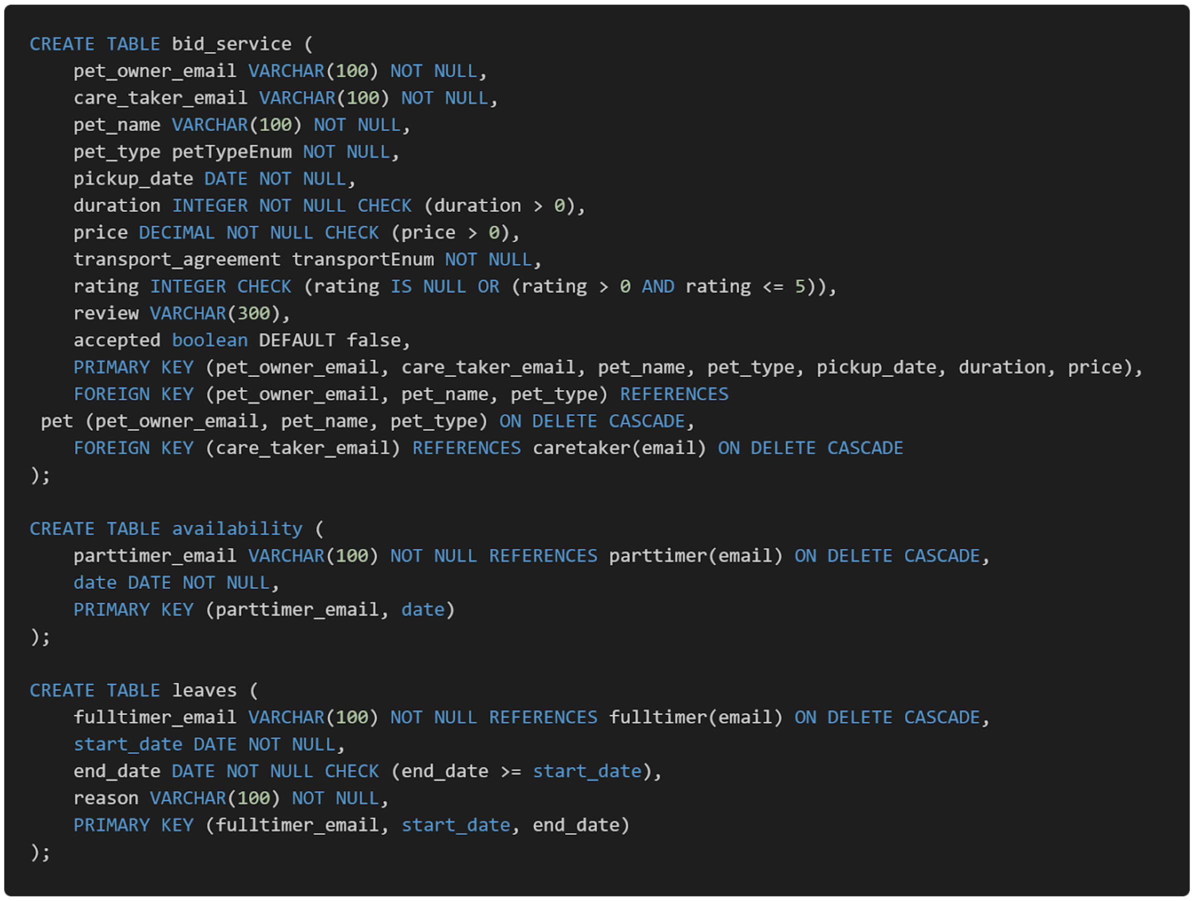
25) A leave record cannot be generated if the dates overlap with an accepted service

26) There can only be a maximum of 5 ongoing service for full time caretakers and part timers if their average rating is >= 4. Else, only a maximum of 2 is allowed.

27) Full timer's bid will be automatically accepted whenever possible

2.4 Relational Schema

```
CREATE TABLE petowner (
    email VARCHAR(100) NOT NULL PRIMARY KEY,
    password VARCHAR(100) NOT NULL,
    name varchar(200) NOT NULL,
    address VARCHAR(200) NOT NULL,
    zone zoneEnum NOT NULL
);

CREATE TABLE caretaker (
    email VARCHAR(100) NOT NULL PRIMARY KEY,
    password VARCHAR(100) NOT NULL,
    name varchar(200) NOT NULL,
    address VARCHAR(200) NOT NULL,
    zone zoneEnum NOT NULL
);

CREATE TABLE PCSAdmin (
    email VARCHAR(100) NOT NULL PRIMARY KEY,
    password VARCHAR(100) NOT NULL,
    name varchar(200) NOT NULL,
    address VARCHAR(200) NOT NULL,
    zone zoneEnum NOT NULL
);

CREATE TABLE parttimer (
    email VARCHAR(100) NOT NULL
    PRIMARY KEY REFERENCES caretaker(email) ON DELETE cascade
);

CREATE TABLE fulltimer (
    email VARCHAR(100) NOT NULL
    PRIMARY KEY REFERENCES caretaker(email) ON DELETE cascade
);

CREATE TABLE creditcard (
    pet_owner_email VARCHAR(100) NOT NULL REFERENCES petowner(email) ON DELETE CASCADE,
    cc_number VARCHAR(20) NOT NULL,
    cvv_code VARCHAR(3) NOT NULL,
    expiry_date DATE NOT NULL,
    PRIMARY KEY (pet_owner_email, cc_number)
);

CREATE TABLE pet (
    pet_owner_email VARCHAR(100) NOT NULL REFERENCES petowner(email) ON DELETE CASCADE,
    pet_name VARCHAR(100) NOT NULL,
    pet_type petTypeEnum NOT NULL,
    special_requirements VARCHAR,
    base_daily_price decimal,
    PRIMARY KEY (pet_owner_email, pet_name, pet_type)
);
```



```sql
CREATE TABLE bid_service (
    pet_owner_email VARCHAR(100) NOT NULL,
    care_taker_email VARCHAR(100) NOT NULL,
    pet_name VARCHAR(100) NOT NULL,
    pet_type petTypeEnum NOT NULL,
    pickup_date DATE NOT NULL,
    duration INTEGER NOT NULL CHECK (duration > 0),
    price DECIMAL NOT NULL CHECK (price > 0),
    transport_agreement transportEnum NOT NULL,
    rating INTEGER CHECK (rating IS NULL OR (rating > 0 AND rating <= 5)),
    review VARCHAR(300),
    accepted boolean DEFAULT false,
    PRIMARY KEY (pet_owner_email, care_taker_email, pet_name, pet_type, pickup_date, duration, price),
    FOREIGN KEY (pet_owner_email, pet_name, pet_type) REFERENCES
 pet (pet_owner_email, pet_name, pet_type) ON DELETE CASCADE,
    FOREIGN KEY (care_taker_email) REFERENCES caretaker(email) ON DELETE CASCADE
);

CREATE TABLE availability (
    parttimer_email VARCHAR(100) NOT NULL REFERENCES parttimer(email) ON DELETE CASCADE,
    date DATE NOT NULL,
    PRIMARY KEY (parttimer_email, date)
);

CREATE TABLE leaves (
    fulltimer_email VARCHAR(100) NOT NULL REFERENCES fulltimer(email) ON DELETE CASCADE,
    start_date DATE NOT NULL,
    end_date DATE NOT NULL CHECK (end_date >= start_date),
    reason VARCHAR(100) NOT NULL,
    PRIMARY KEY (fulltimer_email, start_date, end_date)
);
```

Data Constraints not captured in schema:

2) Users can be a PetOwner, a CareTaker or a PCSAdmin. A user can both be a PetOwner as well as a CareTaker at once.

11) Each Caretaker is either a PartTimer or FullTimer.

23) Each rating and review for each specific transaction can only be done after the service end date

24) Each full timer must have a minimum of 2 x 150 consecutive days of work a year

25) A leave record cannot be generated if the dates overlap with an existing service

26) There can only be a maximum of 5 ongoing service for full time caretakers and part timers if their average rating is >= 4. Else, only a maximum of 2 is allowed.

3 3NF/BCNF (from compiled.sql)

All our schemas are in BCNF as all functional dependencies are either trivial or the attributes on the LHS are key/super keys of the schemas.

Tables T = {**petowner, caretaker, PCSAdmin, parttimer, fulltimer, availability, leaves**} all have only one primary key, so all the tables in T are in BCNF as for all the functional dependencies, *a* -> *A*, in T, a is a superkey. Thus all tables in T are in BCNF, and by Lemma 3, in 3NF.

For tables G = {**creditcard, pet, bid_service**} :

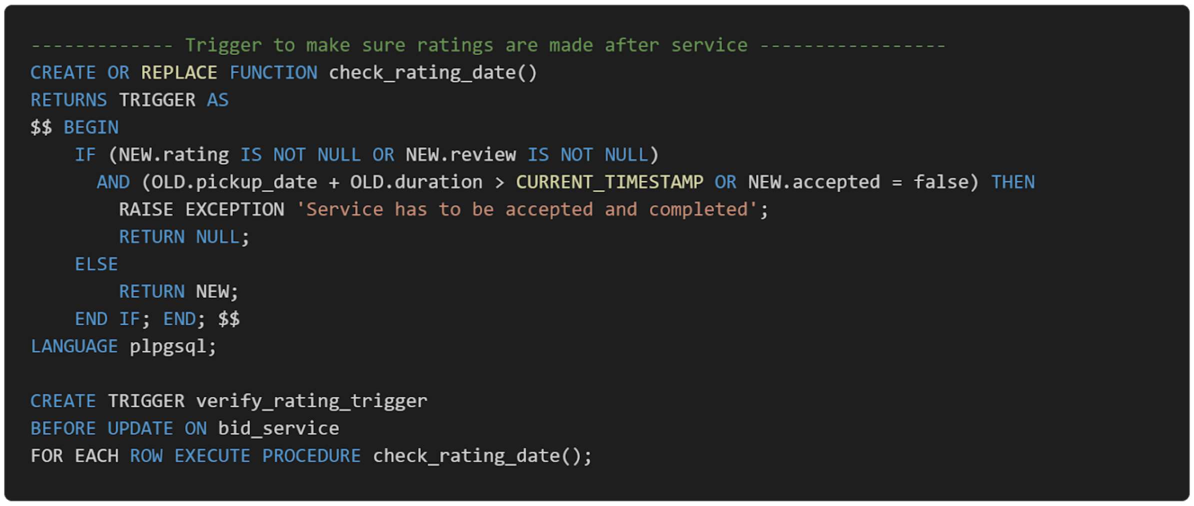
Looking at **creditcard** table, it is in BCNF as all functional dependencies F = {*cc_number, pet_owner_email* -> *cvv_code, cc_number, pet_owner_email* -> *expiry_date*} have the left hand side as the superkey (i*.e. cc_number and pet_owner_email are primary keys)*.

Looking at **pet** table, it is also in BCNF as all the functional dependencies G = {*pet_owner_email, pet_name, pet_type* -> *special_requirements, pet_owner_email, pet_name, pet_type* -> *base_daily_price* } have {*pet_owner_email, pet_name, pet_type*}, the superkey of the **pets** table, on the left hand side. Therefore the **pets** table is in BCNF.

Looking at **bid_service** table, it is in BCNF as all functional dependencies H = { *pet_owner_email, care_taker_email, pet_name, pet_type, pickup_date, duration, price* -> *transport_agreement, pet_owner_email, care_taker_email, pet_name, pet_type, pickup_date, duration, price* -> *rating, pet_owner_email, care_taker_email, pet_name, pet_type, pickup_date, duration, price* -> *review, pet_owner_email, care_taker_email, pet_name, pet_type, pickup_date, duration, price* -> *accepted*} have the superkey {pet_owner_email, care_taker_email, pet_name, pet_type, pickup_date, duration, price} on the left hand side, thus **bid_service** table is in BCNF.

4 Non-Trivial/Interesting Triggers

1) Trigger to ensure that ratings are made after completion of service



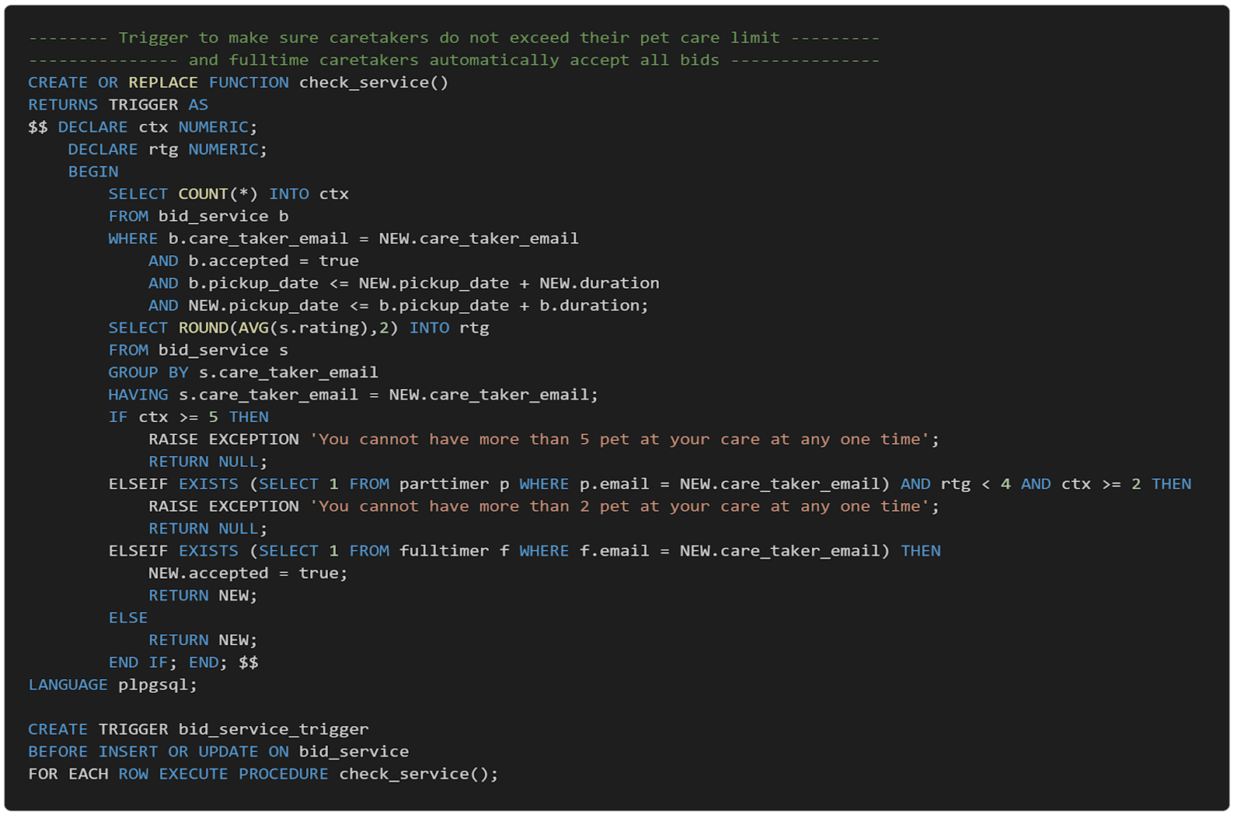
```
------------- Trigger to make sure ratings are made after service -----------------
CREATE OR REPLACE FUNCTION check_rating_date()
RETURNS TRIGGER AS
$$ BEGIN
    IF (NEW.rating IS NOT NULL OR NEW.review IS NOT NULL)
      AND (OLD.pickup_date + OLD.duration > CURRENT_TIMESTAMP OR NEW.accepted = false) THEN
        RAISE EXCEPTION 'Service has to be accepted and completed';
        RETURN NULL;
    ELSE
        RETURN NEW;
    END IF; END; $$
LANGUAGE plpgsql;

CREATE TRIGGER verify_rating_trigger
BEFORE UPDATE ON bid_service
FOR EACH ROW EXECUTE PROCEDURE check_rating_date();
```

Constraints Enforced:

23) Each rating and review for each specific transaction can only be done after the service end date

This trigger is done before every update of an existing record in the bid_service table. First we check that the rating and review column for the new input is not empty and not null, then we check that the total duration of the service is completed by ensuring that the pickup time + the duration of the service stated as per record is less than the current time of updating the record. Also most importantly, we check that the new record was an accepted service. With all these conditions fulfilled, we then allow the service to be updated. By no means should a service have a rating and a review record when the service is not accepted, pending or ongoing, else an exception will be thrown to the users.

2) Trigger to ensure the care limit is not exceeded and allow for automatic acceptance of bid for full timers



```
-------- Trigger to make sure caretakers do not exceed their pet care limit ---------
--------------- and fulltime caretakers automatically accept all bids ---------------
CREATE OR REPLACE FUNCTION check_service()
RETURNS TRIGGER AS
$$ DECLARE ctx NUMERIC;
    DECLARE rtg NUMERIC;
    BEGIN
        SELECT COUNT(*) INTO ctx
        FROM bid_service b
        WHERE b.care_taker_email = NEW.care_taker_email
            AND b.accepted = true
            AND b.pickup_date <= NEW.pickup_date + NEW.duration
            AND NEW.pickup_date <= b.pickup_date + b.duration;
        SELECT ROUND(AVG(s.rating),2) INTO rtg
        FROM bid_service s
        GROUP BY s.care_taker_email
        HAVING s.care_taker_email = NEW.care_taker_email;
        IF ctx >= 5 THEN
            RAISE EXCEPTION 'You cannot have more than 5 pet at your care at any one time';
            RETURN NULL;
        ELSEIF EXISTS (SELECT 1 FROM parttimer p WHERE p.email = NEW.care_taker_email) AND rtg < 4 AND ctx >= 2 THEN
            RAISE EXCEPTION 'You cannot have more than 2 pet at your care at any one time';
            RETURN NULL;
        ELSEIF EXISTS (SELECT 1 FROM fulltimer f WHERE f.email = NEW.care_taker_email) THEN
            NEW.accepted = true;
            RETURN NEW;
        ELSE
            RETURN NEW;
        END IF; END; $$
LANGUAGE plpgsql;

CREATE TRIGGER bid_service_trigger
BEFORE INSERT OR UPDATE ON bid_service
FOR EACH ROW EXECUTE PROCEDURE check_service();
```

Constraints Enforced:

26) There can only be a maximum of 5 ongoing service for full time caretakers and part timers if their average rating is >= 4. Else, only a maximum of 2 is allowed.

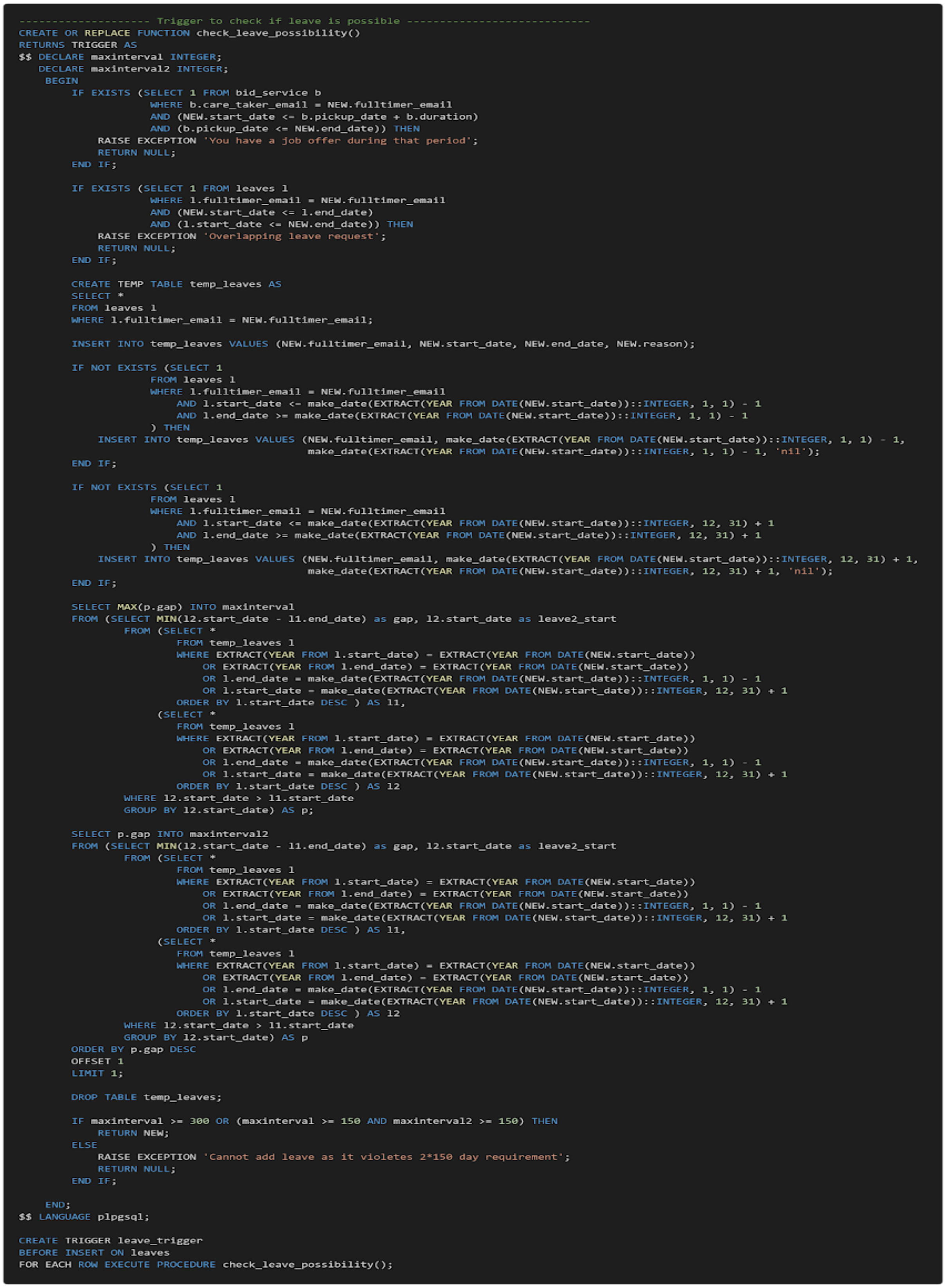
27) Bids are automatically accepted for full time caretakers provided that they are available and have not reached the quota.

This trigger will be invoked before every insertion into the bid_service table. 2 variables ctx, which will count the number of pets they have in their care at the moment, and rtg, which will track their current average ratings, have been initialized. Firstly we will count the total number of existing records for a particular caretaker which have an ongoing service by ensuring that the pickup date of the new record is less than the end date of the existing records and the pick up date of the existing records are less than the end date of the new records. This will be the overlap.

Afterwhich, we will calculate the average rating of that caretaker by using the AVG function of plpgsql.

With the 2 data, we will then compare that if ctx > 5 or if rtg is <4 and ctx > 2, the system will throw an exception and not allow the record to be added as this will violate our constraints.

3) Trigger to ensure that leaves are taken appropriately



```sql
-------------------- Trigger to check if leave is possible ---------------------------
CREATE OR REPLACE FUNCTION check_leave_possibility()
RETURNS TRIGGER AS
$$ DECLARE maxinterval INTEGER;
   DECLARE maxinterval2 INTEGER;
    BEGIN
        IF EXISTS (SELECT 1 FROM bid_service b
                    WHERE b.care_taker_email = NEW.fulltimer_email
                    AND (NEW.start_date <= b.pickup_date + b.duration)
                    AND (b.pickup_date <= NEW.end_date)) THEN
            RAISE EXCEPTION 'You have a job offer during that period';
            RETURN NULL;
        END IF;

        IF EXISTS (SELECT 1 FROM leaves l
                    WHERE l.fulltimer_email = NEW.fulltimer_email
                    AND (NEW.start_date <= l.end_date)
                    AND (l.start_date <= NEW.end_date)) THEN
            RAISE EXCEPTION 'Overlapping leave request';
            RETURN NULL;
        END IF;

        CREATE TEMP TABLE temp_leaves AS
        SELECT *
        FROM leaves l
        WHERE l.fulltimer_email = NEW.fulltimer_email;

        INSERT INTO temp_leaves VALUES (NEW.fulltimer_email, NEW.start_date, NEW.end_date, NEW.reason);

        IF NOT EXISTS (SELECT 1
                    FROM leaves l
                    WHERE l.fulltimer_email = NEW.fulltimer_email
                        AND l.start_date <= make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1
                        AND l.end_date >= make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1
                    ) THEN
            INSERT INTO temp_leaves VALUES (NEW.fulltimer_email, make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1,
                                            make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1, 'nil');
        END IF;

        IF NOT EXISTS (SELECT 1
                    FROM leaves l
                    WHERE l.fulltimer_email = NEW.fulltimer_email
                        AND l.start_date <= make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1
                        AND l.end_date >= make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1
                    ) THEN
            INSERT INTO temp_leaves VALUES (NEW.fulltimer_email, make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1,
                                            make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1, 'nil');
        END IF;

        SELECT MAX(p.gap) INTO maxinterval
        FROM (SELECT MIN(l2.start_date - l1.end_date) as gap, l2.start_date as leave2_start
                FROM (SELECT *
                        FROM temp_leaves l
                        WHERE EXTRACT(YEAR FROM l.start_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR EXTRACT(YEAR FROM l.end_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR l.end_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1
                            OR l.start_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1
                        ORDER BY l.start_date DESC ) AS l1,
                     (SELECT *
                        FROM temp_leaves l
                        WHERE EXTRACT(YEAR FROM l.start_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR EXTRACT(YEAR FROM l.end_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR l.end_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1
                            OR l.start_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1
                        ORDER BY l.start_date DESC ) AS l2
                WHERE l2.start_date > l1.start_date
                GROUP BY l2.start_date) AS p;

        SELECT p.gap INTO maxinterval2
        FROM (SELECT MIN(l2.start_date - l1.end_date) as gap, l2.start_date as leave2_start
                FROM (SELECT *
                        FROM temp_leaves l
                        WHERE EXTRACT(YEAR FROM l.start_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR EXTRACT(YEAR FROM l.end_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR l.end_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1
                            OR l.start_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1
                        ORDER BY l.start_date DESC ) AS l1,
                     (SELECT *
                        FROM temp_leaves l
                        WHERE EXTRACT(YEAR FROM l.start_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR EXTRACT(YEAR FROM l.end_date) = EXTRACT(YEAR FROM DATE(NEW.start_date))
                            OR l.end_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 1, 1) - 1
                            OR l.start_date = make_date(EXTRACT(YEAR FROM DATE(NEW.start_date))::INTEGER, 12, 31) + 1
                        ORDER BY l.start_date DESC ) AS l2
                WHERE l2.start_date > l1.start_date
                GROUP BY l2.start_date) AS p
        ORDER BY p.gap DESC
        OFFSET 1
        LIMIT 1;

        DROP TABLE temp_leaves;

        IF maxinterval >= 300 OR (maxinterval >= 150 AND maxinterval2 >= 150) THEN
            RETURN NEW;
        ELSE
            RAISE EXCEPTION 'Cannot add leave as it violetes 2*150 day requirement';
            RETURN NULL;
        END IF;

    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER leave_trigger
BEFORE INSERT ON leaves
FOR EACH ROW EXECUTE PROCEDURE check_leave_possibility();
```

Constraints Enforced:

24) Each full timer must have a minimum of 2 x 150 consecutive days of work a year

25) A leave record cannot be generated if the dates overlap with an existing service.

This trigger is executed before an insertion is done to the leaves table. First we check to see if there is an existing job offer for the caretaker by ensuring that the new record's leave start date is less than the service end date and the service pick up date is less than the new record's end date. If this condition is not fulfilled, an exception will be thrown.

Else, a temporary table called "temp_leaves" is created to get all the leaves of the specific fulltimer. A dummy leave value will be inserted in the temporary table if an existing leave does not exist at the start and end of the year. Afterwhich, we will insert the new leave record into the temporary table and get all the minimum intervals between 2 leave records in temp_leaves tables and find the maximum interval from that. If the highest interval is >= 300 or the highest and second highest are both => 150, we will insert the new record. Else an exception will be thrown.

5 SQL Code

1)

```
SELECT
    CASE
        WHEN p.email IS NULL THEN a.name
        WHEN a.email IS NULL THEN p.name
    END AS name,
    CASE
        WHEN p.email IS NULL THEN a.email
        WHEN a.email IS NULL THEN p.email
    END AS email,
    CASE
        WHEN p.email IS NULL THEN a.zone
        WHEN a.email IS NULL THEN p.zone
    END AS userzone,
    CASE
        WHEN p.email IS NULL THEN 'PCSAdmin'
        WHEN a.email IS NULL AND f.email IS NULL THEN 'Part Timer'
        WHEN a.email IS NULL AND t.email IS NULL THEN 'Full Timer'
    END accounttype
FROM caretaker c NATURAL JOIN petowner p
    FULL OUTER JOIN fulltimer f ON f.email = c.email
    FULL OUTER JOIN parttimer t ON t.email = c.email
    FULL OUTER JOIN PCSAdmin a ON a.email = p.email
WHERE (p.email = $1 AND p.password = $2)
    OR (a.email = $1 AND a.password = $2)
```

This query is to get the relevant data of the users of Pet Society based on the input email and password. This query will join all tables of caretaker, petowner, fulltimer, part timer and the admin. The query will ensure that all caretaker is labelled as either part timer or fulltimer. Furthermore, since there is an overlap constraint over caretakers and petowner, a natural join is used for the 2 tables, while an outer join is used for the other tables where null will be used to fill the columns for relations where the condition does not meet.

2)

```sql
SELECT c.email, c.name, c.address, c.zone , r.average
FROM caretaker c INNER JOIN parttimer f ON c.email = f.email
    LEFT OUTER JOIN (SELECT ROUND(AVG(s.rating),2) AS average,
                            s.care_taker_email AS email
                        FROM bid_service s
                        GROUP BY s.care_taker_email) AS r
        ON r.email = c.email
WHERE (SELECT COUNT(*) FROM availability a
        WHERE a.parttimer_email = c.email
            AND a.date >= DATE($1)
            AND a.date <= DATE($1) + $2::INTEGER -1) = $2
    AND c.email <> $3
ORDER BY r.average DESC;
```

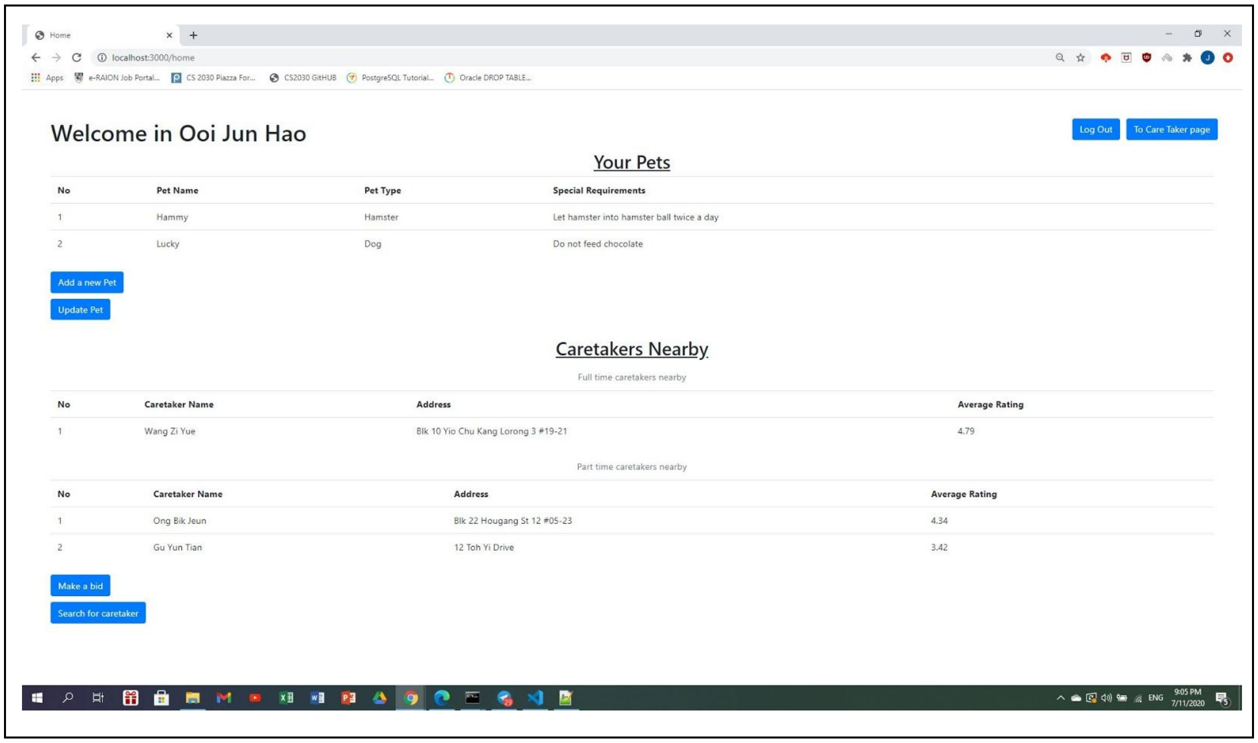
This query is specific for part timers. This will retrieve all part timers who have availabilities during the stated service period requested by the pet owners. This will then allow the owners to bid for the part timers they want. The user will input 2 information, $1 - start date of the service, $2 - total duration of the service. $3 - will be the email of the petowner himself. To find out which caretakers are part time caretakers, we use an inner join between the caretaker table and the part timer table. Also, the average rating of each part timer will be displayed for the pet owner to view as the final table will be ordered from highest rating to the least. The condition placed will count all the availabilities that lie inside the requested service period. If the count that lies in the requested service period equals the service period, that would mean that the part timer has an availability for everyday during that service period and hence would potentially be able to accept the service request. The final condition in the WHERE clause is to ensure that the pet owners do not bid for themselves as there is an overlapping constraint for caretakers and pet owners

3)

```sql
SELECT p.no_of_jobs AS no_of_jobs, to_char(to_timestamp (p.month::text, 'MM'), 'Month')
AS month
FROM (
    SELECT COUNT(*) AS no_of_jobs, EXTRACT(MONTH FROM b.pickup_date) AS month
    FROM bid_service b
    WHERE b.accepted = true
    GROUP BY EXTRACT(MONTH FROM b.pickup_date)
    ) AS p
WHERE p.no_of_jobs >= ALL(
                        SELECT COUNT(*) AS no_of_jobs
                        FROM bid_service b
                        WHERE b.accepted = true
                        GROUP BY EXTRACT(MONTH FROM b.pickup_date));
```

Table p will count the total number of jobs taken up for each month and this query will retrieve the month with the highest number of jobs accepted and completed for the year. The WHERE clause will evaluate to true if the highest number of jobs accepted from p is greater than or equal to all the values returned by the subquery.

6 Application in action



Welcome in Ooi Jun Hao

Your Pets

No	Pet Name	Pet Type	Special Requirements
1	Hammy	Hamster	Let hamster into hamster ball twice a day
2	Lucky	Dog	Do not feed chocolate

Caretakers Nearby

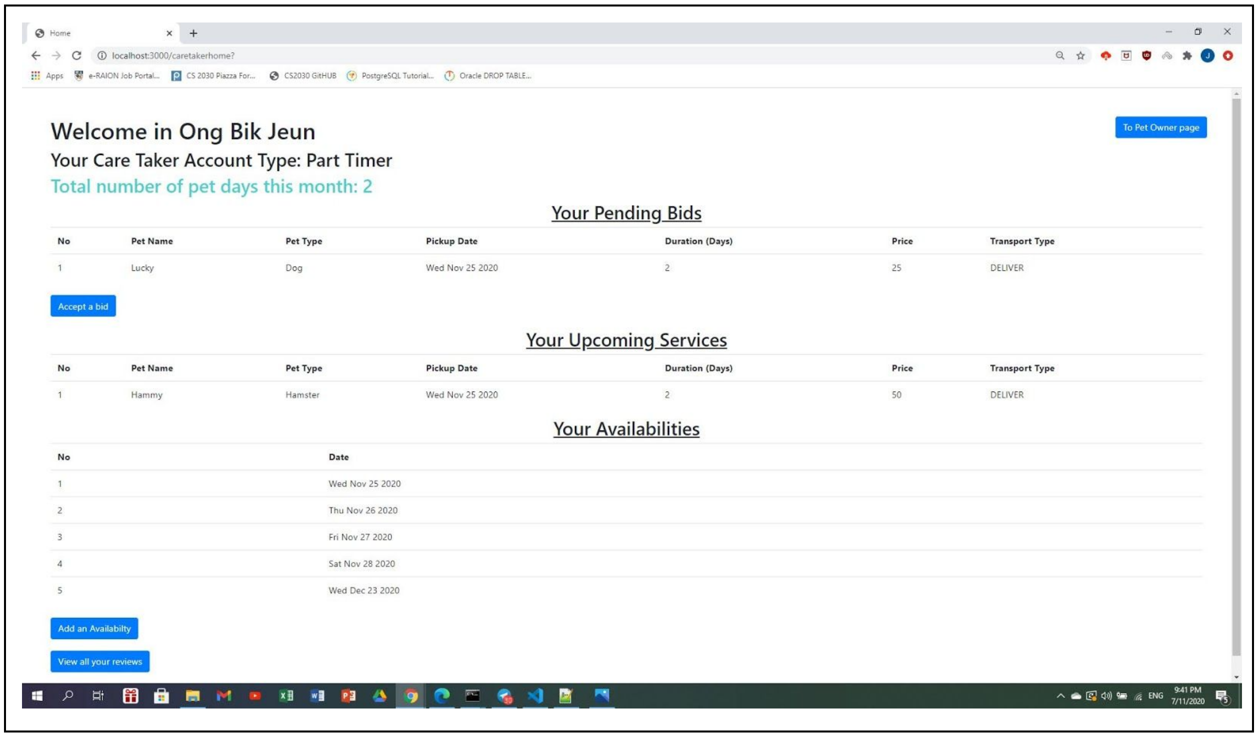
Full time caretakers nearby

No	Caretaker Name	Address	Average Rating
1	Wang Zi Yue	Blk 10 Yio Chu Kang Lorong 3 #19-21	4.79

Part time caretakers nearby

No	Caretaker Name	Address	Average Rating
1	Ong Bik Jeun	Blk 22 Hougang St 12 #05-23	4.34
2	Gu Yun Tian	12 Toh Yi Drive	3.42

PetOwner Home Page



Welcome in Ong Bik Jeun

Your Care Taker Account Type: Part Timer

Total number of pet days this month: 2

Your Pending Bids

No	Pet Name	Pet Type	Pickup Date	Duration (Days)	Price	Transport Type
1	Lucky	Dog	Wed Nov 25 2020	2	25	DELIVER

Your Upcoming Services

No	Pet Name	Pet Type	Pickup Date	Duration (Days)	Price	Transport Type
1	Hammy	Hamster	Wed Nov 25 2020	2	50	DELIVER

Your Availabilities

No	Date
1	Wed Nov 25 2020
2	Thu Nov 26 2020
3	Fri Nov 27 2020
4	Sat Nov 28 2020
5	Wed Dec 23 2020

CareTaker Home Page

7 Summary and Lessons Learnt

In conclusion, our web app will allow users to create accounts, add new pet data, bid for services and record all services done. On our website, bids will be decided by the caretakers themselves or by the system in the case of full timers. Once a bid is successful, the service will be carried out and reviews and ratings will be provided which in turn will determine the capabilities of the caretakers.

This project has been an eye opener for everyone as it gave us the opportunity to learn new tech stacks which were never explored before such as postgreSQL, nodeJS as well as bootstrap. Prior to this project, our knowledge of SQL is limited to queries alone and we only had a rough idea of real world instances that SQL could apply to. This project has allowed us to be able to see how SQL is applied in real world applications, through simulating an actual deployment of SQL queries alongside frontend web development. Even though the project is still small scaled and arguably frontend development is not what we learned in this module, the reconciliation of frontend and backend knowledge in this project has allowed us to fully experience the potential and versatility of SQL.

Another difficulty we had faced in this project also includes troubleshooting in an unfamiliar setting. As we came in with mostly zero knowledge and hands-on experience with nodeJS and bootstrap, this meant that we were bound to and had faced many difficulties when troubleshooting our application. Each error we encounter would result in another round of troubleshooting to search for solutions online. However, as we progressed further we realised that we started to solve some errors on our own and we learned that this difficulty comes with every new tech stack or programming language, where one would have to learn it the hard way through effort and trial and error, before one could become proficient in it.

All in all, we learned that while the project may have been a pain to us all, we are glad that we had not been simply spoon fed with knowledge in this module, as it has truly brought about much learning experiences to each and every one of us.