CS2102 Project Report

AY 2020/2021 Semester 1

Group 43

Khoo Kai Xin Cassie E0200704

Jeremy Chua Yong Siang E0201344

Swa Yong Shen E0406833

Lee Xuan Wei, Jeremy E0425730

Koh Huai Ze E0426227

1. Project Responsibilities

Khoo Kai Xin Cassie
- PCS Admin Dashboard
- PCS Admin Profile
- View Caretaker Summary Info
- Manage Users
- Manage Pet Types
- Search Transactions

Jeremy Chua Yong Siang
- Search caretaker page for pet owner
- Prebid page

Swa Yong Shen
- Creation of all accounts (admin, caretaker, pet owner)
- Bid for caretaker page
- Edit bid for caretaker page
- Payment page

Lee Xuan Wei, Jeremy
- Pet Owner Dashboard and CRUD of Pet Owner Edit Particulars
- CRUD of a Pet Owner’s Pets and Pet Profile
- Caretaker Profile (for Pet Owner to view)
- CRUD of Review

Koh Huai Ze
- Caretaker Dashboard
- Caretaker Edit Particulars
- Apply Leave
- Caretaker Pet Types Overview, Add Pet Type

2. Data Requirements and Application Functionalities

2.1 Data Requirements

1. Every Account is identified by its email. Its password and name should also be recorded.
2. Every Account is either a pcs_admin or a User.
3. Every User has a location and an address.There are two types of Users -- pet_owner and care_taker.
4. pet_owner may register a credit card.
5. Each pet is identified by a pet_name and pet_owner. They can have only one pet_owner. Their special_requirement can also be recorded if applicable.
6. Each pet_type is identified by a name and its base_daily_price should also be recorded.
7. Each pet is of one pet_type.
8. Different pets can have the same pet_type.
9. The monthly_pet_days and monthly_salary of the care_taker should be computed then updated after every hire where the hire_status is completed.
10. The rating and max_concurrent_pet_limit for the care_taker will be updated when a rating is given for a successful hire.
11. The bank_account of a care_taker should be recorded.
12. A care_taker may take care of more than one pet_type of pets, and the daily_price they charge for each pet_type should be recorded.
13. Every care_taker is either a part_timer or a full_timer.
14. Each part_timer indicates their availability, which is identified by a start_date, end_date and the email address of the part_timer.
15. Each part_timer can indicate multiple periods of availability, and we do not need to track the availability if the part_timer deletes his account.
16. Each leave should be identified by a start_date, end_date and the email address of the full_timer.
17. Each full_timer can have multiple leaves, and we do not need to track the leaves if the full_timer deletes his account.
18. pet_owner(s) can hire care_taker(s) to care of their pet(s) over a date_range.
19. Each date_range is identified by the start_date and end_date.
20. For each new hire, the transaction_date, hire_status, num_pet_days, method_of_payment, method_of_pet_transfer and total_cost should be recorded.
21. Each pet_owner can choose to give review_text and/or rating for each completed hire.

2.2 Application Functionalities

Common

1. New users can sign up as pet owners or part time caretakers.
2. Users can login and logout of the system.
3. Unauthenticated users can see the top 4 rated caretakers and also 4 recently completed transactions.
4. Unauthenticated users can view caretaker profiles.

Pet Owner

1. Once logged in, pet owners are directed to their dashboard, where they have an overview of their particulars, their pets, the top caretakers in their location and past transactions.
2. Pet owners can add/ edit their pets, edit their particulars or hire for caretakers.
3. Pet owners can search for caretakers.
4. Pet owners can make payment for their hire using cash or credit card.
5. Pet owners can give a rating and make a review after every completed hire.

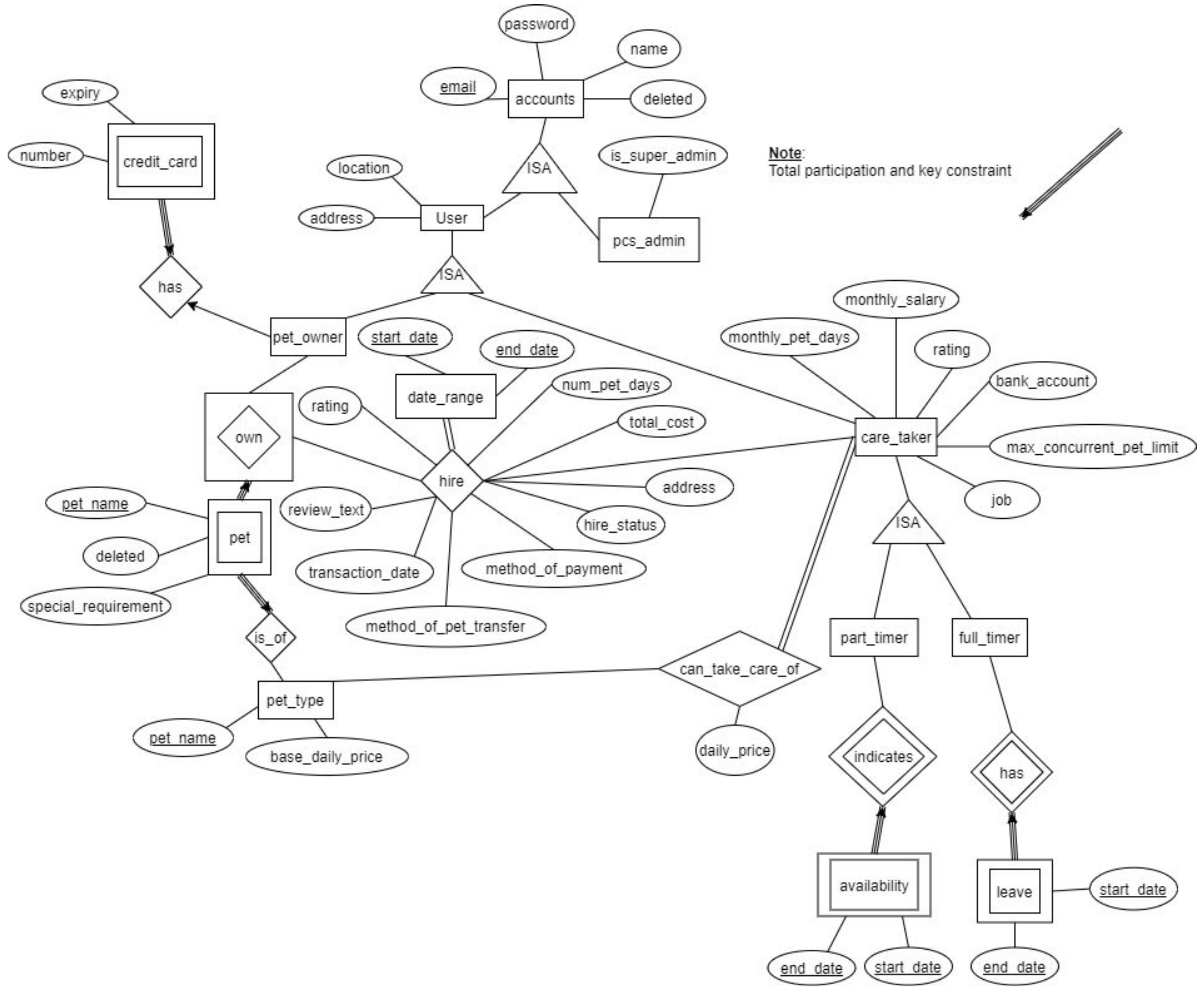
Caretaker

1. Similarly, caretakers are directed to their dashboard, where they can see an overview of their particulars, the types of pets they can take care of, their leave requests and their past transactions.
2. Caretakers can edit their particulars and add/delete pet types that they can take care of.
3. Full-timers can submit new leave requests while part-timers can indicate the range of dates they are available to accept jobs.
4. Part-time caretakers can choose to accept/reject hires made by pet owners.
5. If the method of payment is cash, caretakers can indicate that the payment has been received.
6. Caretakers can indicate that they have finished caring for a pet.

PCS Admin

1. Upon logging in, PCS admins are directed to the PCS Admin Dashboard where they can view summary statistics such as the number of pets taken care of, the salary to be paid and the number of active transactions this month.
2. The PCS Admin Dashboard also displays a bar graph plotting the number of transactions made over time for both part time and full time caretakers. There is also a pie chart showing the distribution of the different methods of pet transfers specified in the transactions.
3. PCS Admins can view their profile and edit their name and/or password.
4. PCS Admins can view all pet types supported by the platform and edit the base daily price for each pet type.
5. PCS Admins may also add pet types for the platform.
6. PCS Admins can search transactions and filter them by transaction status and/or whether they are transactions made in the current month
7. PCS Admins can choose to activate and deactivate caretaker and pet owner accounts
8. PCS Admins can add new full time caretakers
9. PCS Super Admin can perform tasks available to PCS Admin. On top of that, they will be able to activate, deactivate and add new PCS Admin accounts

3. ER Diagram



Constraints not captured by ER Diagram:

1. Caretakers must fulfil 2 blocks of 150 consecutive days in a year.
2. Range of dates of leaves cannot overlap.
3. Each pet can only be in 1 transaction that is in progress at a time.
4. A pet owner can only hire a caretaker who is not on leave (for full-timer) or who is available (for part-timer).
5. A pet owner cannot hire a caretaker who has already exceeded his max concurrent pet limit on any day.
6. The start date for each hire, availability and leave must be before or on the same day as the end date.
7. The transaction date for each hire must be before or on the same day as the start date.
8. The number of days of hire should be more than 0.
9. The max concurrent pet limit of a full-timer is fixed to 5 while for a part-timer it varies depending on his average rating.
10. Pet transfer methods can only be 'cPickup', 'oDeliver' or 'office'.

11. Method of payment can only be ‘cash’ or ‘credit card’.
12. Hire status can only be ‘pendingAccept’, ‘rejected’, ‘pendingPayment’, ‘paymentMade’, ‘inProgress’ or ‘completed’.
13. Each account is either pcs_admin or user but not both (covering constraint, no overlap constraint).
14. Each user is either care_taker or pet_owner but not both (covering constraint, no overlap constraint).
15. A pet owner can only hire a caretaker who can take care of pet types matching his pets.
16. The rating of a caretaker is the average rating of all his completed hires with ratings given by pet owners.
17. The daily price charged by each individual caretaker is scaled with the rating of a caretaker.

4. Relational Schema

```
CREATE TABLE pcs_admin(
  email VARCHAR PRIMARY KEY,
  name VARCHAR NOT NULL,
  password VARCHAR NOT NULL,
  is_super_admin BOOLEAN NOT NULL DEFAULT FALSE,
  deleted BOOLEAN NOT NULL DEFAULT FALSE
);

CREATE TABLE pet_owner(
  email VARCHAR PRIMARY KEY,
  name VARCHAR NOT NULL,
  password VARCHAR NOT NULL,
  location VARCHAR NOT NULL,
  address VARCHAR,
  deleted BOOLEAN NOT NULL DEFAULT FALSE
);

CREATE TABLE has_credit_card(
  number VARCHAR NOT NULL,
  email VARCHAR REFERENCES pet_owner(email) PRIMARY KEY,
  expiry VARCHAR NOT NULL
);

CREATE TYPE job_type AS ENUM ('part_timer', 'full_timer');

CREATE TABLE care_taker(
  email VARCHAR PRIMARY KEY,
  name VARCHAR NOT NULL,
  password VARCHAR NOT NULL,
  location VARCHAR NOT NULL,
  monthly_pet_days INTEGER DEFAULT 0,
  monthly_salary NUMERIC,
  rating NUMERIC,
  bank_account VARCHAR,
  max_concurrent_pet_limit INTEGER,
  job job_type NOT NULL,
  address VARCHAR,
  deleted BOOLEAN NOT NULL DEFAULT FALSE,
  CHECK (job = 'full_timer' AND max_concurrent_pet_limit = 5)
);

CREATE VIEW accounts AS (
  SELECT email, name, password, deleted, 1 AS type FROM pet_owner
  UNION
  SELECT email, name, password, deleted, 2 AS type FROM care_taker
  UNION
  SELECT email, name, password, deleted, 0 AS type FROM pcs_admin
);

CREATE TABLE part_timer(
  email VARCHAR PRIMARY KEY REFERENCES care_taker(email)
);

CREATE TABLE full_timer(
  email VARCHAR PRIMARY KEY REFERENCES care_taker(email)
);

CREATE TABLE own_pet (
  pet_name VARCHAR NOT NULL,
  special_requirement VARCHAR NOT NULL,
  email VARCHAR REFERENCES pet_owner(email),
  deleted BOOLEAN NOT NULL DEFAULT false,
  PRIMARY KEY(pet_name, email)
);

CREATE TABLE pet_type (
  name VARCHAR PRIMARY KEY,
  base_daily_price NUMERIC NOT NULL
);

CREATE TABLE can_take_care_of(
  email VARCHAR REFERENCES care_taker(email),
  daily_price NUMERIC NOT NULL,
  pet_type VARCHAR REFERENCES pet_type(name),
  PRIMARY KEY(email, pet_type)
);

CREATE TABLE is_of (
  pet_type VARCHAR REFERENCES pet_type(name),
  pet_name VARCHAR NOT NULL,
  owner_email VARCHAR NOT NULL,
  FOREIGN KEY (pet_name, owner_email) REFERENCES own_pet(pet_name, email),
  PRIMARY KEY (pet_name, owner_email)
);

CREATE TABLE date_range (
  start_date DATE,
  end_date DATE,
  PRIMARY KEY(start_date, end_date)
);

CREATE TYPE hire_status AS ENUM('pendingAccept', 'rejected', 'pendingPayment',
'paymentMade', 'inProgress', 'completed', 'cancelled');

CREATE TYPE pet_transfer AS ENUM('cPickup', 'oDeliver', 'office');

CREATE TYPE method_of_payment AS ENUM('cash', 'creditcard');

CREATE TABLE hire (
  owner_email VARCHAR,
  pet_name VARCHAR,
  ct_email VARCHAR REFERENCES care_taker(email),
  num_pet_days INTEGER NOT NULL CHECK (num_pet_days > 0),
  total_cost NUMERIC NOT NULL,
  hire_status hire_status NOT NULL,
  method_of_pet_transfer pet_transfer NOT NULL,
  method_of_payment method_of_payment NULL,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  transaction_date DATE NOT NULL,
  rating INTEGER CHECK (rating >= 1 AND rating <= 5),
  review_text VARCHAR,
  address VARCHAR,
  PRIMARY KEY(owner_email, pet_name, ct_email, start_date, end_date),
  FOREIGN KEY (owner_email, pet_name) REFERENCES own_pet(email, pet_name),
  FOREIGN KEY(start_date, end_date) REFERENCES date_range(start_date, end_date),
  CHECK (start_date <= end_date),
  CHECK (transaction_date <= start_date)
);

CREATE TABLE indicates_availability (
  email VARCHAR NOT NULL REFERENCES part_timer(email),
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  PRIMARY KEY(email, start_date, end_date),
  CHECK (start_date <= end_date)
);

CREATE TABLE has_leave (
  email VARCHAR NOT NULL REFERENCES full_timer(email),
  start_date DATE NOT NULL,
  end_date DATE NOT NULL,
  PRIMARY KEY(email, start_date, end_date),
  CHECK (start_date <= end_date)
);
```

Constraints not captured by relational schema:

1. Caretakers must fulfil 2 blocks of 150 consecutive days in a year.
2. Range of dates of leave applications cannot overlap.
3. Each pet can only be in 1 transaction that is in progress at a time.
4. A pet owner can only hire a caretaker who is not on leave (for full-timer) or who is available (for part-timer).
5. A pet owner cannot hire a caretaker who has already exceeded his max concurrent pet limit on any day.
6. Each account is either pcs_admin or user but not both (covering constraint, no overlap constraint).
7. Each user is either care_taker or pet_owner but not both (covering constraint, no overlap constraint).
8. A pet owner can only hire a caretaker who can take care of pet types matching his pets.
9. The rating of a caretaker is the average rating of all his completed hires with ratings given by pet owners.
10. The daily price charged by each individual caretaker is scaled with his/her average rating.

5. Normal Forms

For each table in our database Ri, if the attributes are A,B,C,D and the primary key is A,B, the projection of functional dependencies, F[Ri] = {AB -> CD}. Since AB is a superkey, Ri is in BCNF. Since every table in our database has a primary key and in addition,since our application constraints do not introduce any additional functional dependencies, every table is in BCNF and hence the database is in BCNF.

6. Triggers

Trigger 1

Checks that the caretaker has not already met the max concurrent pet limit for any date between the start and end date of a new hire.

1. Generates all dates between the start and end date of a new hire into a table all_dates.
2. For each one_date in all_dates, count the number of accepted and not yet completed hires for the caretaker that have the one_date in between the start and end date of the hire.
3. Check that for all one_date in all_dates, the number of hires counted in (2) is less than the max_concurrent_pet_limit of the caretaker involved in the hire.

```
CREATE OR REPLACE FUNCTION add_hire() RETURNS TRIGGER AS
$$
BEGIN
  IF (EXISTS(
  SELECT 1
  FROM (select one_date::date from generate_series(NEW.start_date, NEW.end_date, '1
day'::interval) one_date) all_dates, hire
  WHERE hire.ct_email = NEW.ct_email
    AND hire.hire_status <> 'rejected'
    AND hire.hire_status <> 'completed'
    AND hire.hire_status <> 'pendingAccept'
    AND hire.hire_status <> 'cancelled'
    AND hire.start_date <= all_dates.one_date
    AND hire.end_date >= all_dates.one_date
  GROUP BY all_dates.one_date
  HAVING COUNT(*) > (SELECT max_concurrent_pet_limit
                     FROM care_taker
                     WHERE email = NEW.ct_email)
  )) THEN
    RAISE NOTICE 'Exceed max concurrent';
    RETURN NULL;
  END IF;
  RETURN NEW;
END;
```

```
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS hire_add_hire ON pet_care.hire;

CREATE TRIGGER hire_add_hire BEFORE INSERT ON hire FOR EACH ROW EXECUTE
PROCEDURE add_hire();
```

Trigger 2
When a pet owner adds a rating for a caretaker, query to get the average rating for that caretaker. Update the caretaker's rating in the care_taker table accordingly.
Then, adjust the prices the caretaker can charge based on his rating, scaling according to the base_daily_price set by the PCS admin.
If the caretaker is a part timer and his rating is more than 2, then we adjust his pet limit accordingly.

```
CREATE OR REPLACE FUNCTION increase_rating_and_price() RETURNS TRIGGER AS
$$
DECLARE total_trxn NUMERIC;
DECLARE total_rating NUMERIC;
DECLARE base_price NUMERIC;
DECLARE job_ct VARCHAR;
DECLARE avg_rating NUMERIC;
BEGIN
 SELECT INTO total_trxn, total_rating COUNT(H1.rating), SUM(H1.rating)
  FROM hire H1
  WHERE H1.ct_email = NEW.ct_email
  AND (H1.rating IS NOT NULL OR H1.rating <> 0)
  AND H1.hire_status = 'completed';
 UPDATE care_taker
  SET rating =
   CASE
    WHEN total_trxn = 0 THEN 0
    ELSE total_rating/total_trxn
   END
  WHERE email = NEW.ct_email;

 UPDATE can_take_care_of C
  SET daily_price =
   CASE
    WHEN total_trxn = 0 THEN P.base_daily_price
    ELSE P.base_daily_price * (1 + (total_rating/total_trxn)/5)
   END
  FROM pet_type P
```

```
    WHERE P.name = C.pet_type
    AND C.email = NEW.ct_email;

  SELECT INTO job_ct, avg_rating job, rating
    FROM care_taker
    WHERE care_taker.email = NEW.ct_email;

  IF job_ct = 'part_timer' AND avg_rating > 2 THEN
    UPDATE care_taker
      SET max_concurrent_pet_limit = FLOOR(avg_rating)
    WHERE email = NEW.ct_email;
  END IF;

  RETURN NEW;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS increase_rating_and_price ON pet_care.hire;

CREATE TRIGGER increase_rating_and_price AFTER UPDATE ON hire FOR EACH ROW
EXECUTE PROCEDURE increase_rating_and_price();
```

Trigger 3

This trigger checks for every hire that the pet type of the pet in the hire is of a type which can be taken care of by the caretaker. Allows the procedure to proceed if the caretaker can take care of the pet type, and rolls back otherwise.

```
CREATE OR REPLACE FUNCTION check_can_take_care_of() RETURNS TRIGGER AS
$$
BEGIN
  IF (
    (SELECT pet_type FROM is_of I WHERE I.owner_email = NEW.owner_email AND
I.pet_name = NEW.pet_name)
    IN
    (SELECT pet_type FROM can_take_care_of WHERE email = NEW.ct_email)
  ) THEN
    RETURN NEW;
  END IF;
  RETURN NULL;
END;
$$
LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS hire_can_take_care_of ON pet_care.hire;

CREATE TRIGGER hire_can_take_care_of BEFORE INSERT ON hire FOR EACH ROW
EXECUTE PROCEDURE check_can_take_care_of();
```

7. Complex and Interesting Queries

Query 1

This query gets all the caretakers who are in the location selected by the user, available in the date range selected by the user and can take care of the pets listed by the user by combining 1 and 2 and taking away 3.

1. Get all part time caretakers based on the user selected start date and end date in the range of the availability selected by the part time caretaker and the user selected location equals the location of the part time caretaker and the user selected pet types is a superset of the pet types that the part time caretaker can take care of.
2. Get all full time caretakers based on the user selected location equals the location of the full time caretaker and the user selected pet types is a superset of the pet types that the full time caretaker can take care of.
3. Get all full time caretakers based on the user selected start date and end date in the range of the leave applied by the full time caretaker and the user selected location equals the location of the full time caretaker and the user selected pet types is a superset of the pet types that the full time caretaker can take care of.

```
(SELECT c.email, c.name, c.location, c.rating, c.job, t.daily_price, ARRAY_AGG(t.daily_price + p.base_daily_price) AS price, ARRAY_AGG(t.pet_type) AS pet_types
  FROM care_taker c
  INNER JOIN can_take_care_of t ON c.email = t.email
  INNER JOIN pet_type p ON t.pet_type = p.name
  INNER JOIN indicates_availability as a ON c.email = a.email
  WHERE a.start_date <= $1
    AND a.end_date >= $2
    AND c.location = $3
    AND NOT EXISTS
      (SELECT UNNEST($4::varchar[]) AS pet_type
      EXCEPT
      SELECT pet_type
        FROM can_take_care_of t1
        WHERE t1.email = t.email)
    AND c.deleted = FALSE
  GROUP BY c.email, t.daily_price)
UNION
(SELECT c.email, c.name, c.location, c.rating, c.job, t.daily_price, ARRAY_AGG(t.daily_price + p.base_daily_price) AS price, ARRAY_AGG(t.pet_type) AS pet_types
  FROM care_taker c
  INNER JOIN can_take_care_of t ON c.email = t.email
  INNER JOIN pet_type p ON t.pet_type = p.name
  WHERE c.job = 'full_timer'
    AND c.location = $3
    AND NOT EXISTS
      (SELECT UNNEST($4::varchar[]) AS pet_type
      EXCEPT
      SELECT pet_type
        FROM can_take_care_of t1
        WHERE t1.email = t.email)
    AND c.deleted = FALSE
  GROUP BY c.email, t.daily_price
EXCEPT
SELECT c.email, c.name, c.location, c.rating, c.job, t.daily_price, ARRAY_AGG(t.daily_price +
p.base_daily_price) AS price, ARRAY_AGG(t.pet_type) AS pet_types
  FROM care_taker c
  INNER JOIN can_take_care_of t ON c.email = t.email
  INNER JOIN pet_type p ON t.pet_type = p.name
  INNER JOIN has_leave as h ON c.email = h.email
  WHERE c.job = 'full_timer'
    AND (h.start_date > $1 AND h.start_date < $2)
    OR (h.end_date > $1 AND h.end_date < $2)
    AND c.location = $3
    AND NOT EXISTS
      (SELECT UNNEST($4::varchar[]) AS pet_type
      EXCEPT
      SELECT pet_type
        FROM can_take_care_of t1
        WHERE t1.email = t.email)
    AND c.deleted = FALSE
  GROUP BY c.email, t.daily_price)
ORDER BY rating DESC NULLS LAST, daily_price DESC NULLS LAST;
```

Query 2

This query first checks that the hire is recently completed, before adding the number of pet days completed to the monthly_pet_days count for the caretaker involved.
Then, if the caretaker is a part timer, we update his salary by 75% of the total cost of the hire.
If the caretaker is a full timer, we split into 2 cases:

1. If the monthly_pet_days is already more than or equal to 60, we add 80% of the total cost directly to the caretaker's salary.
2. If not, we add 80% of the cost of the excess number of pet days above 60 from this hire to the salary of the caretaker.

```
CREATE OR REPLACE FUNCTION update_monthly_stats() RETURNS TRIGGER AS
$$
DECLARE old_monthly_pet_days NUMERIC;
DECLARE old_salary NUMERIC;
DECLARE job job_type;
BEGIN
SELECT INTO old_monthly_pet_days, old_salary, job monthly_pet_days, monthly_salary, job
FROM care_taker WHERE email = NEW.ct_email;
IF (OLD.hire_status = 'inProgress' AND NEW.hire_status = 'completed') THEN
  UPDATE care_taker SET monthly_pet_days = old_monthly_pet_days + NEW.num_pet_days
  WHERE email = NEW.ct_email;

  IF (job = 'part_timer') THEN
    UPDATE care_taker SET monthly_salary = old_salary + 0.75 * NEW.total_cost
    WHERE email = NEW.ct_email;
  ELSE
    IF (old_monthly_pet_days >= 60) THEN
      UPDATE care_taker SET monthly_salary = old_salary + 0.8 * NEW.total_cost
      WHERE email = NEW.ct_email;
    ELSE
      IF (old_monthly_pet_days + NEW.num_pet_days > 60) THEN
        UPDATE care_taker SET monthly_salary = old_salary +
        (old_monthly_pet_days + NEW.num_pet_days - 60) / NEW.num_pet_days *
NEW.total_cost * 0.8
        WHERE email = NEW.ct_email;
      END IF;
    END IF;
  END IF;
END IF;
RETURN NEW;
END;
$$
LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_monthly_stats ON pet_care.hire;

CREATE TRIGGER update_monthly_stats AFTER UPDATE ON hire FOR EACH ROW
EXECUTE PROCEDURE update_monthly_stats();
```

Query 3
Count the number of transactions that occur in each month and year pair for the part time and full time caretakers in the last 12 months.

1. Count the number of transactions that occur in each month and year pair for the part time caretakers and order them such that the rows are in descending year and month. Combine the “month” and “year” columns into a “date” column.
2. Repeat (1) for the full time caretakers and combine the results.
3. Obtain data for the last 12 months by only taking the first 12 rows.

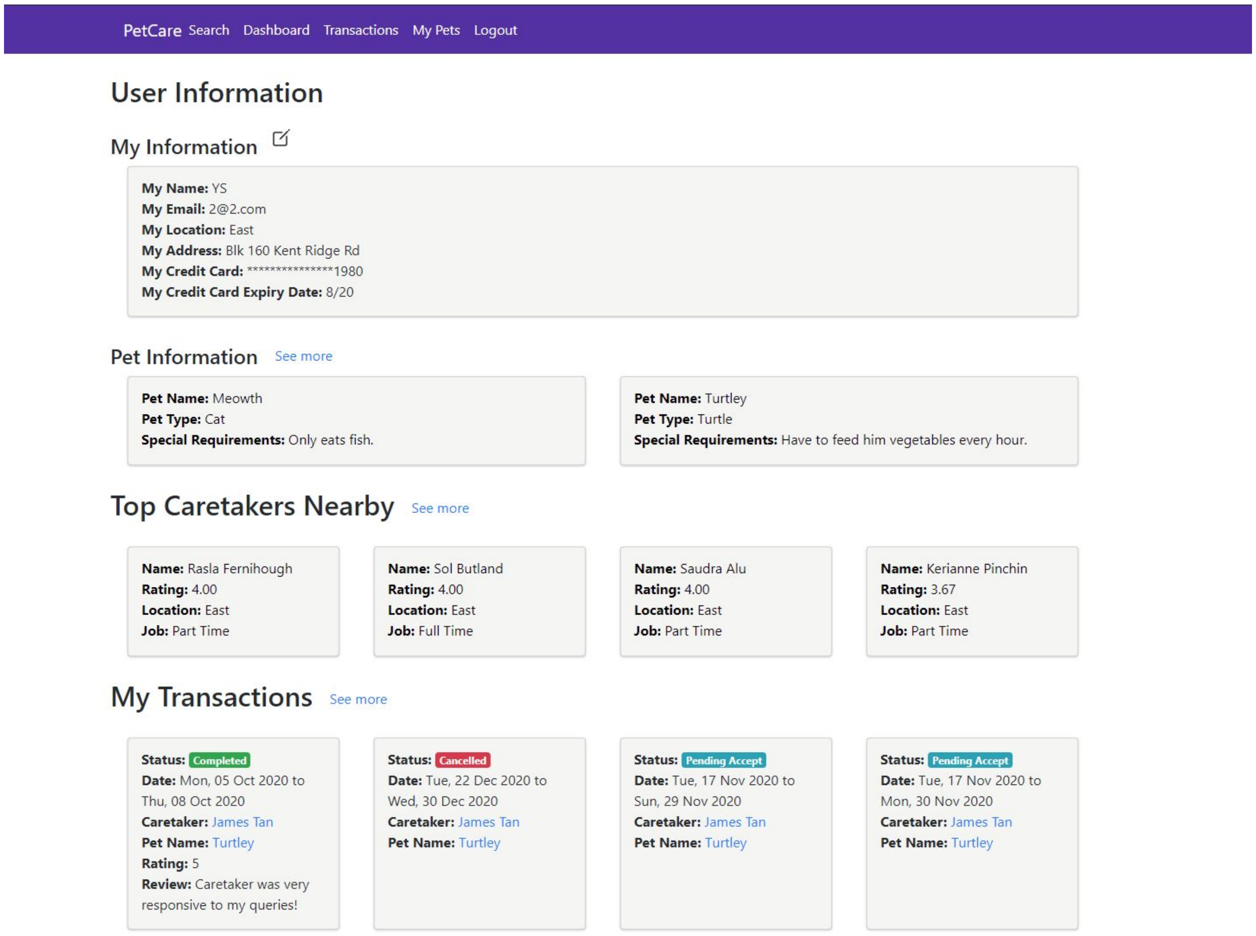
```
SELECT *
  FROM (SELECT concat(concat(month, '/'), year) as date, count_PT
                FROM (SELECT date_part('month', transaction_date) AS month, date_part('year',
transaction_date) AS year, COUNT(transaction_date) AS count_PT
                                FROM hire
                                WHERE ct_email IN (SELECT PT.email FROM part_timer PT)
                                GROUP BY date_part('month', transaction_date), date_part('year',
transaction_date)
                                ORDER BY year desc, month desc) as derivedtable) as table1
    FULL JOIN (SELECT concat(concat(month, '/'), year) as date, count_FT
                          FROM (SELECT date_part('month', transaction_date) AS month,
date_part('year', transaction_date) AS year, COUNT(transaction_date) AS count_FT
                                          FROM hire
                                          WHERE ct_email IN (SELECT FT.email FROM full_timer FT) GROUP
BY date_part('month', transaction_date), date_part('year', transaction_date) ORDER BY year
desc, month desc) as derivedtable2) AS table2 on table1.date = table2.date LIMIT 12;
```

8. Specification of Tech Stack

1. NodeJS
2. HTML
3. CSS
4. JQuery
5. EJS
6. PostgreSQL
7. Bootstrap
8. Heroku

9. Application Screenshots

Dashboard for a Pet Owner



User Information

My Information

My Name: YS
My Email: 2@2.com
My Location: East
My Address: Blk 160 Kent Ridge Rd
My Credit Card: ***************1980
My Credit Card Expiry Date: 8/20

Pet Information See more

Pet Name: Meowth
Pet Type: Cat
Special Requirements: Only eats fish.

Pet Name: Turtley
Pet Type: Turtle
Special Requirements: Have to feed him vegetables every hour.

Top Caretakers Nearby See more

Name: Rasla Fernihough
Rating: 4.00
Location: East
Job: Part Time

Name: Sol Butland
Rating: 4.00
Location: East
Job: Full Time

Name: Saudra Alu
Rating: 4.00
Location: East
Job: Part Time

Name: Kerianne Pinchin
Rating: 3.67
Location: East
Job: Part Time

My Transactions See more

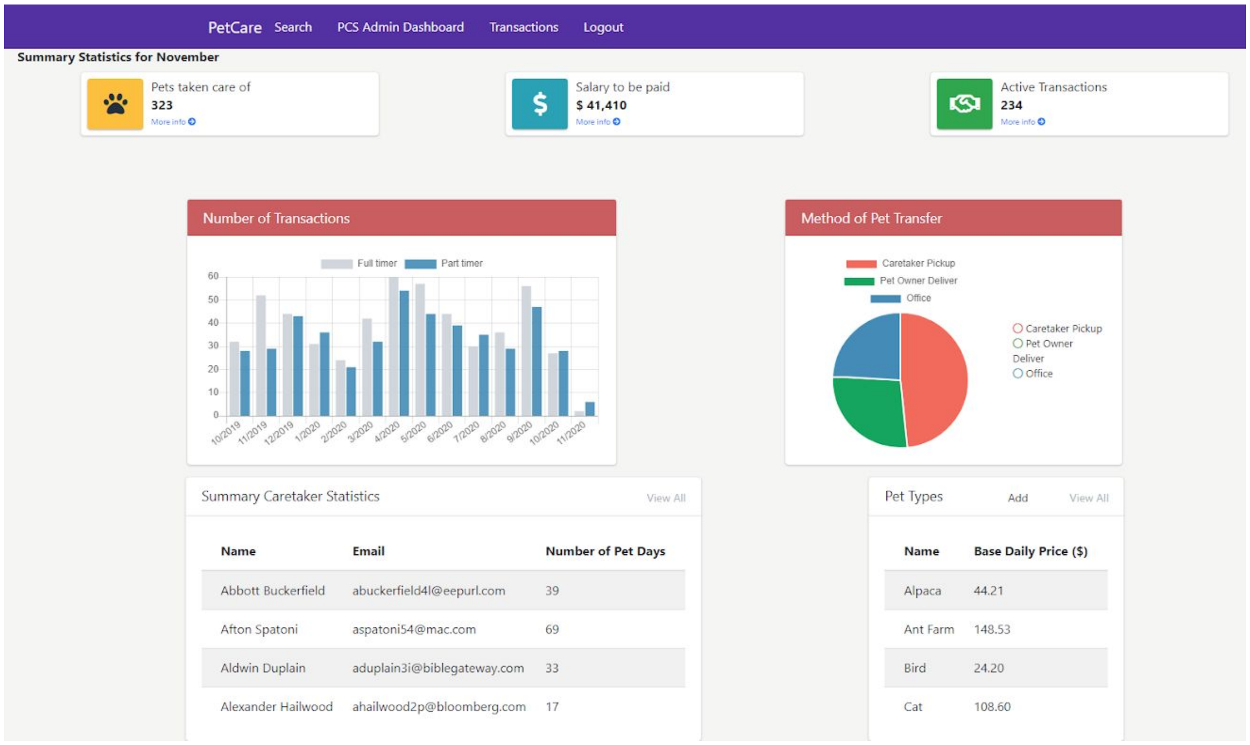
Status: Completed
Date: Mon, 05 Oct 2020 to Thu, 08 Oct 2020
Caretaker: James Tan
Pet Name: Turtley
Rating: 5
Review: Caretaker was very responsive to my queries!

Status: Cancelled
Date: Tue, 22 Dec 2020 to Wed, 30 Dec 2020
Caretaker: James Tan
Pet Name: Turtley

Status: Pending Accept
Date: Tue, 17 Nov 2020 to Sun, 29 Nov 2020
Caretaker: James Tan
Pet Name: Turtley

Status: Pending Accept
Date: Tue, 17 Nov 2020 to Mon, 30 Nov 2020
Caretaker: James Tan
Pet Name: Turtley

PCS Admin Summary Dashboard



PetCare Search PCS Admin Dashboard Transactions Logout

Summary Statistics for November

Pets taken care of	323
Salary to be paid	$ 41,410
Active Transactions	234

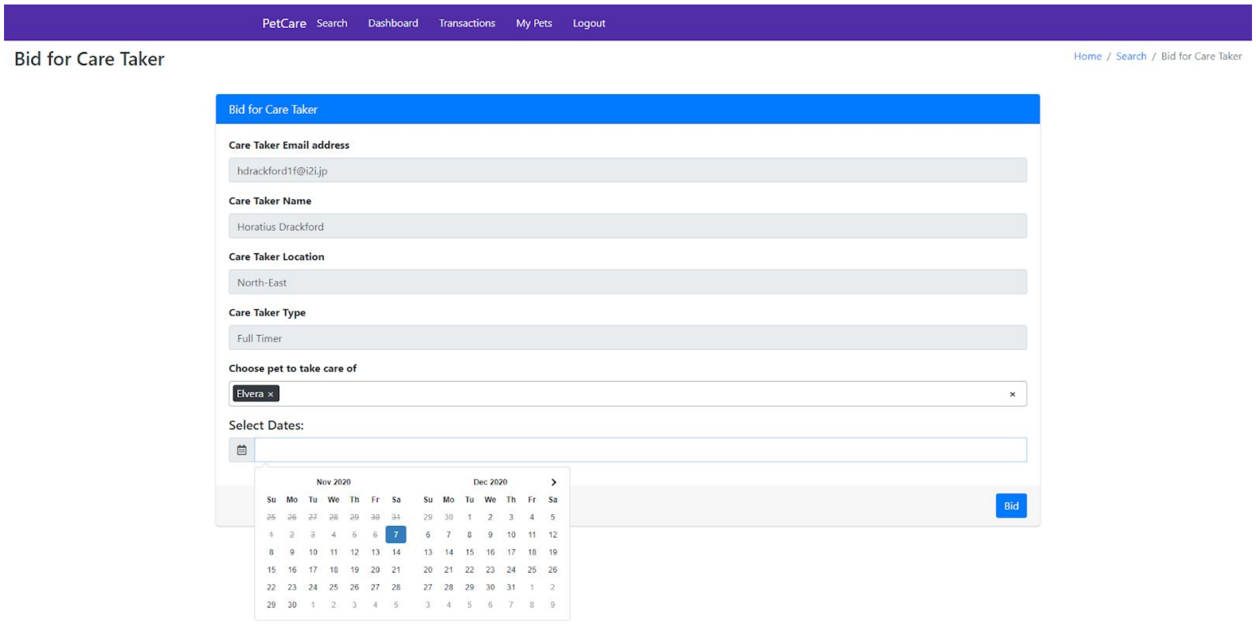
Summary Caretaker Statistics View All

Name	Email	Number of Pet Days
Abbott Buckerfield	abuckerfield4l@eepurl.com	39
Afton Spatoni	aspatoni54@mac.com	69
Aldwin Duplain	aduplain3i@biblegateway.com	33
Alexander Hailwood	ahailwood2p@bloomberg.com	17

Pet Types Add View All

Name	Base Daily Price ($)
Alpaca	44.21
Ant Farm	148.53
Bird	24.20
Cat	108.60

Bid page



PetCare Search Dashboard Transactions My Pets Logout

Bid for Care Taker

Home / Search / Bid for Care Taker

Bid for Care Taker

Care Taker Email address	hdrackford1f@i2i.jp
Care Taker Name	Horatius Drackford
Care Taker Location	North-East
Care Taker Type	Full Timer
Choose pet to take care of	Elvera ×

Select Dates:

Nov 2020

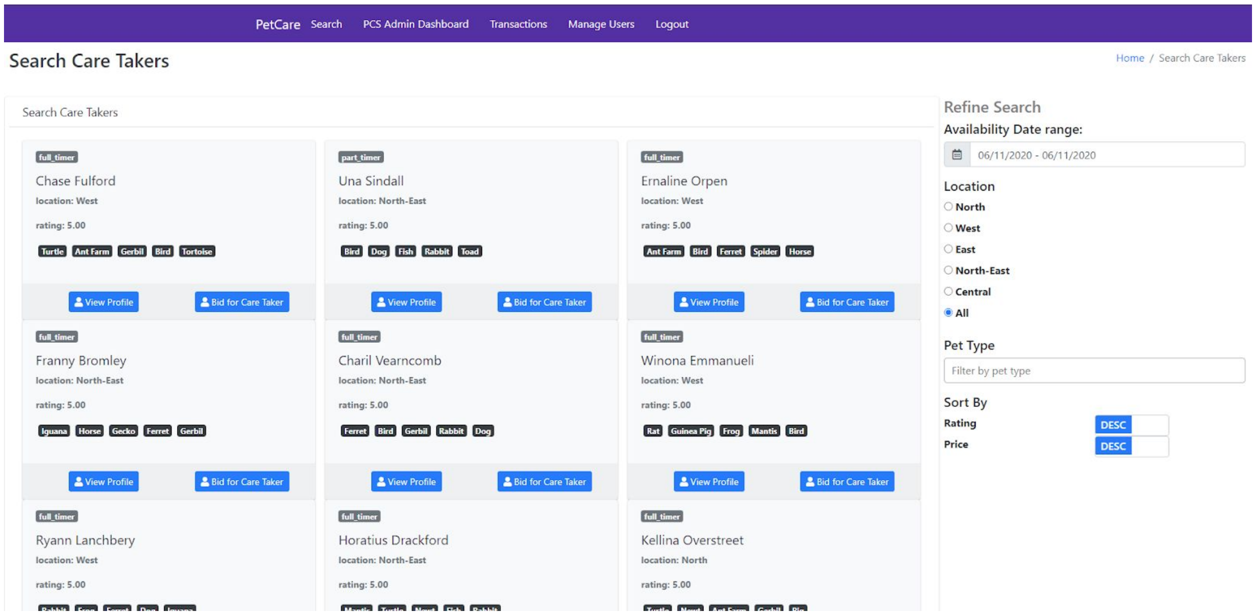
Su	Mo	Tu	We	Th	Fr	Sa
~~25~~	~~26~~	~~27~~	~~28~~	~~29~~	~~30~~	~~31~~
~~1~~	~~2~~	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	1	2	3	4	5

Dec 2020

Su	Mo	Tu	We	Th	Fr	Sa
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

Bid

Search page



PetCare Search PCS Admin Dashboard Transactions Manage Users Logout

Search Care Takers

Home / Search Care Takers

Search Care Takers

full_timer
Chase Fulford
location: West
rating: 5.00
Turtle Ant Farm Gerbil Bird Tortoise
View Profile | Bid for Care Taker

part_timer
Una Sindall
location: North-East
rating: 5.00
Bird Dog Fish Rabbit Toad
View Profile | Bid for Care Taker

full_timer
Ernaline Orpen
location: West
rating: 5.00
Ant Farm Bird Ferret Spider Horse
View Profile | Bid for Care Taker

full_timer
Franny Bromley
location: North-East
rating: 5.00
Iguana Horse Gecko Ferret Gerbil
View Profile | Bid for Care Taker

full_timer
Charil Vearncomb
location: North-East
rating: 5.00
Ferret Bird Gerbil Rabbit Dog
View Profile | Bid for Care Taker

full_timer
Winona Emmanueli
location: West
rating: 5.00
Rat Guinea Pig Frog Mantis Bird
View Profile | Bid for Care Taker

full_timer
Ryann Lanchbery
location: West
rating: 5.00

full_timer
Horatius Drackford
location: North-East
rating: 5.00

full_timer
Kellina Overstreet
location: North
rating: 5.00

Refine Search

Availability Date range:
06/11/2020 - 06/11/2020

Location
North
West
East
North-East
Central
[✓ All]

Pet Type
Filter by pet type

Sort By
Rating DESC
Price DESC

10. Project Reflections

Difficulties:

1) Picking up new technologies
    - Some of us had to learn part of the technological stack such as NodeJS from scratch in order to implement our application.
    - We also had to learn how to utilise HTML and CSS to design a decent user interface which fits the data that we wanted to display.
    - Also, we had to figure out how to host our project online using Heroku
2) Need to standardise when merging individual components together
    - We had to ensure that different components of the system worked well together and that the flow of the system was coherent.
3) Large amounts of coordination and discussion required
    - We often had to spend many hours in our meetings discussing how features should be designed and providing feedback to one another.
4) Data initialization might not always fulfil our constraints and changes to our sql tables required changes to the data.

Lessons learnt:

1) There needs to be proper planning of ER diagrams and database schemas so that no major changes have to be made after we start to develop our application
2) We realised that functional dependencies help to reduce some data redundancies in our database.
3) It is important for our ER diagrams and SQL schema to capture application constraints/business requirements to reduce ambiguity.
4) It is good to research on good practices done by others online before programming a functionality, to get more inspiration and exposure to different methods of approaching a problem.