CS2102 Database Systems
Project Report
Team 16

Name of Student: Ang Lu Shing
Matric no.: A0187364U

Name of Student: Cheyanne Sim
Matric no.: A0191171L

Name of Student: Deon Chung Hui
Matric no.: A0205227N

Name of Student: Ng Jun Rong Terence
Matric no.: A0184003U

Name of Student: Ng Zi Xin
Matric no.: A0189083U

Name	**Responsibilities**
Ang Lu Shing	Pet, Bid
Cheyanne Sim	Pet Owner
Deon Chung Hui	Care Taker
Ng Jun Rong Terence	Login, Sign up, UI
Ng Zi Xin	PCS Admin

Table of Contents

Data requirements

Data	Requirement
username, name, password, area, position, reason, type, rtype, requirement, transfer_method, payment_mode, review	VARCHAR
credit_card, price, base_price, daily_price	NUMERIC
date, start_date, end_date	DATE
phone_number, rating	INTEGER
has_paid, is_completed	BOOLEAN

Functionalities

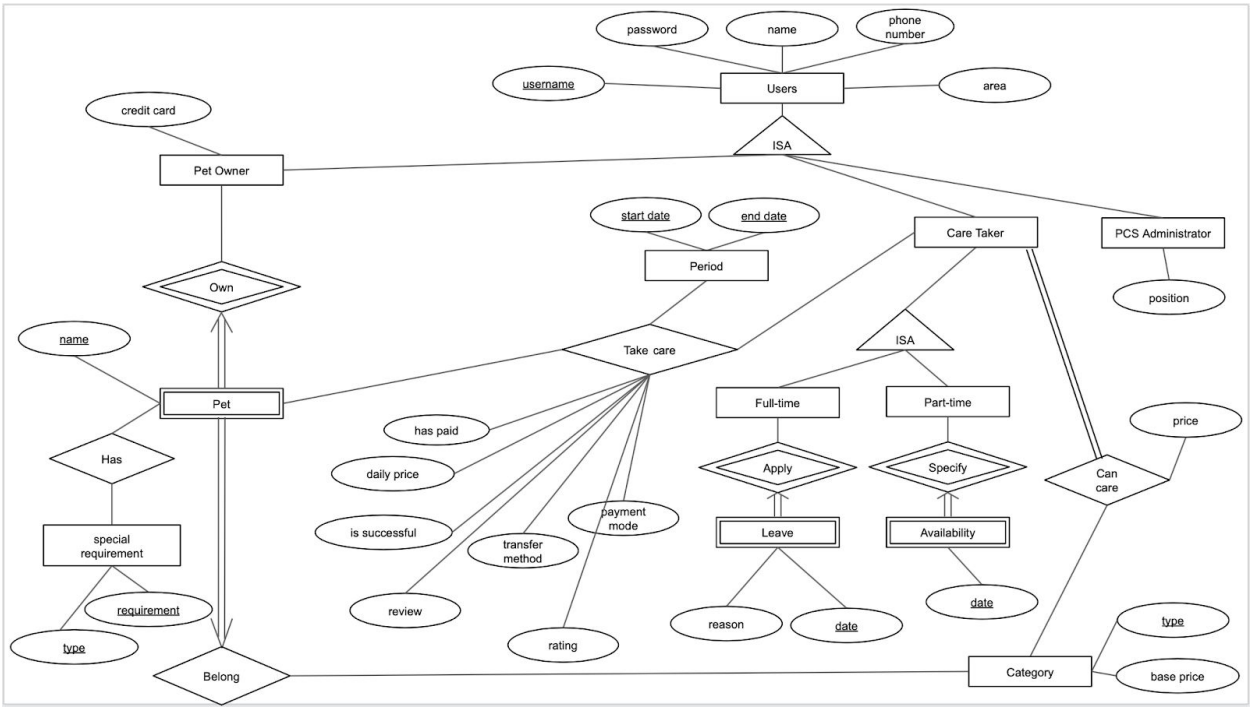
- All users can:
    - View/update/delete their profiles (except for root admin)
    - View base prices
    - View care takers for a type of pet available at given period
- Pet Owners can:
    - Sign up
    - Add/update/delete pets
    - View past and current transactions
    - Rate and review completed transactions
- Care Takers can:
    - Add the type of pets they can care for
    - Set price for type of pets they can care for if their average ratings are 4 and above
    - Specify availability if part timer
    - Apply leave if full timer
    - View summary for current month which includes number of transactions, pets, pet days and salary
    - View past and current transactions
    - Mark a transaction as paid or completed
- PCS Administrators can:
    - View summary of past 12 months which includes number of transactions, pets, pet days, salary, earnings and profit for each month and the maximum across the past 12 months
    - View monthly report with summary for individual care takers, total number of transactions, pets, pet days, salary, earnings and profit
    - Add care taker
    - Add admin

Constraints

- General:

    - There are three kinds of users: Pet Owner, Care Taker, and PCS Administrator (covering constraint).
    - Every user can be identified by their username. Their password, area and phone number must be recorded.
    - Phone numbers should be between 7 and 15 digits.
    - Pet Owners and Care Takers may use the same account (overlapping constraint).
    - A PCS Administrator cannot be a Pet Owner or Care Taker (not overlapping).
    - Each pet category can be identified by the type and the base price must be recorded.
- Pet Owner:
    - A Pet Owner can pre-register his credit card. The credit card number is recorded.
    - Each Pet Owner can own zero or more pets.
    - Each pet belongs to a category, and has special requirements (e.g. daily walk and types of food).
    - Each special requirement is identified by the type and requirement.
    - Each pet is owned by a Pet Owner, and the pet name identifies the pet uniquely from all the pets from the same Pet Owner. If the Pet Owner account is deleted, we do not keep track of his pets anymore.
- Care Taker:
    - A Care Taker can either be a part timer or a full timer (covering but not overlapping).
    - Each Care Taker must be able to care for at least one type of pet.
    - A Care Taker can decide on the daily price for each pet type that he can care for if he has a good rating (4 out of 5), but it should never fall below the base price specified by PCS.
    - Each full time Care Taker can apply for leave indicating the date and an optional reason. If the full time Care Taker's account is deleted, we do not keep track of his leave anymore.
    - A full time Care Taker cannot apply for leave if he has at least one pet under him (no emergency leave) or if he did not work for a minimum of 2 x 150 consecutive days for the year.
    - Each part time Care Taker can specify the dates that they are available for the current year and next year. If the part time Care Taker's account is deleted, we do not keep track of his availability anymore.
- PCS Administrator:
    - Each PCS Administrator has his position in the company recorded.
- Bid:
    - Pet Owner username, pet name, Care Taker username, start date and end date uniquely identify a bid.
    - There should be at most one bid involving the same Pet Owner and pet at any point in time.
    - For each bid, the payment mode (card, cash) and transfer method (Deliver, Pick up, PCS) are recorded.
    - For each bid, the payment success status and daily price are also recorded.
    - A Care Taker must not take care of pets that they cannot care for.
    - A full time Care Taker can take care of up to 5 pets at any point in time.
    - A part time Care Taker can take care of up to 2 pets unless he has a good rating (4 out of 5), but no more than 5.
    - Care Takers always accept the job immediately if possible.
    - A Pet Owner can give up to one rating and one review per transaction.

ER model



Constraints not captured by ER model

- General:
    - There are three kinds of users: Pet Owner, Care Taker, and PCS Administrator (covering constraint).
    - Phone numbers should be between 7 and 15 digits.
    - Pet Owners and Care Takers may use the same account (overlapping constraint).
    - A PCS Administrator cannot be a Pet Owner or Care Taker (not overlapping).
- Care Taker:
    - A Care Taker can either be a part timer or a full timer (covering but not overlapping).
    - A Care Taker can decide on the daily price for each pet type that he can care for if he has a good rating (4 out of 5), but it should never fall below the base price specified by PCS.
    - A full time Care Taker cannot apply for leave if he has at least one pet under him (no emergency leave) or if he did not work for a minimum of 2 x 150 consecutive days for the year.
    - Each part time Care Taker can specify the dates that they are available for the current year and next year.
- Bid:
    - There should be at most one bid involving the same Pet Owner and pet at any point in time.
    - A Care Taker must not take care of pets that they cannot care for.
    - A full time Care Taker can take care of up to 5 pets at any point in time.
    - A part time Care Taker can take care of up to 2 pets unless he has a good rating (4 out of 5), but no more than 5.
    - Care Takers always accept the job immediately if possible.

Relational schema

```
CREATE TABLE category (
  type VARCHAR
     PRIMARY KEY,
  base_price NUMERIC NOT NULL
);

CREATE TABLE users (
  username VARCHAR
     PRIMARY KEY,
  password VARCHAR NOT NULL,
  name VARCHAR NOT NULL,
  phone_number INTEGER NOT NULL,
  area VARCHAR NOT NULL
);

CREATE TABLE pcs_admin (
  username VARCHAR
     PRIMARY KEY
     REFERENCES users(username)
     ON DELETE cascade,
  position VARCHAR NOT NULL
);

CREATE TABLE care_taker (
  username VARCHAR
     PRIMARY KEY
     REFERENCES users(username)
     ON DELETE cascade
);

CREATE TABLE full_time (
  username VARCHAR
     PRIMARY KEY
     REFERENCES care_taker(username)
     ON DELETE cascade
);

CREATE TABLE apply_leave (
  username VARCHAR
     REFERENCES full_time(username)
     ON DELETE cascade,
  date DATE,
  reason VARCHAR,
  PRIMARY KEY (username, date)
);
```

```
CREATE TABLE part_time (
  username VARCHAR
     PRIMARY KEY
     REFERENCES care_taker(username)
     ON DELETE cascade
);

CREATE TABLE specify_availability (
  username VARCHAR
     REFERENCES part_time(username)
     ON DELETE cascade,
  date DATE,
  PRIMARY KEY (username, date)
);

CREATE TABLE can_care (
  username VARCHAR
     REFERENCES care_taker(username)
     ON DELETE cascade,
  type VARCHAR
     REFERENCES category(type),
  price NUMERIC,
  PRIMARY KEY (username, type)
);

CREATE TABLE pet_owner (
  username VARCHAR
     PRIMARY KEY
     REFERENCES users(username)
     ON DELETE cascade,
  credit_card NUMERIC
);

CREATE TABLE own_pet_belong (
  username VARCHAR
     REFERENCES pet_owner(username)
     ON DELETE cascade,
  name VARCHAR,
  type VARCHAR NOT NULL
     REFERENCES category(type),
  PRIMARY KEY (username, name)
);

CREATE TABLE special_requirement (
  rtype VARCHAR,
  requirement VARCHAR,
  PRIMARY KEY (rtype, requirement)
```

```
);

CREATE TABLE has (
  username VARCHAR,
  name VARCHAR,
  rtype VARCHAR,
  requirement VARCHAR,
  FOREIGN KEY (username, name) REFERENCES own_pet_belong(username, name) ON DELETE
cascade,
  FOREIGN KEY (rtype, requirement) REFERENCES special_requirement(rtype,
requirement),
  PRIMARY KEY (username, name, rtype, requirement)
);

CREATE TABLE period (
  start_date DATE,
  end_date DATE,
  CHECK (start_date <= end_date),
  PRIMARY KEY (start_date, end_date)
);

CREATE TABLE take_care (
  username VARCHAR,
  name VARCHAR,
  start_date DATE,
  end_date DATE,
  ctuname VARCHAR REFERENCES care_taker(username)
  ON DELETE CASCADE,
  has_paid BOOLEAN,
  daily_price NUMERIC,
  is_completed BOOLEAN NOT NULL DEFAULT FALSE,
  review VARCHAR,
  rating INTEGER CHECK (rating >= 1 AND rating <= 5),
  transfer_method VARCHAR,
  payment_mode VARCHAR,
  FOREIGN KEY (username, name) REFERENCES own_pet_belong(username, name) ON DELETE
CASCADE,
  FOREIGN KEY (start_date, end_date) REFERENCES period(start_date, end_date),
  PRIMARY KEY (username, name, start_date, end_date, ctuname)
);
```

Constraints not enforced by relational schema

The following constraints are enforced using triggers:

- General:
  - A PCS Administrator cannot be a Pet Owner or Care Taker (not overlapping).
- Care Taker:

    - A Care Taker can decide on the daily price for each pet type that he can care for if he has a good rating (4 out of 5), but it should never fall below the base price specified by PCS.
    - A full time Care Taker cannot apply for leave if he has at least one pet under him (no emergency leave) or if he did not work for a minimum of 2 x 150 consecutive days for the year.
- Bid:
    - There should be at most one bid involving the same Pet Owner and pet at any point in time.
    - A Care Taker must not take care of pets that they cannot care for.
    - A full time Care Taker can take care of up to 5 pets at any point in time.
    - A part time Care Taker can take care of up to 2 pets unless he has a good rating (4 out of 5), but no more than 5.

The following constraints cannot be enforced but were taken into consideration in the design of the application:

- General:
    - There are three kinds of users: Pet Owner, Care Taker, and PCS Administrator (covering constraint).
    - Phone numbers should be between 7 and 15 digits.
    - Pet Owners and Care Takers may use the same account (overlapping constraint).
- Care Taker:
    - A Care Taker can either be a part timer or a full timer (covering but not overlapping).
    - Each Care Taker must be able to care for at least one type of pet.
    - Each part time Care Taker can specify the dates that they are available for the current year and next year.
- Bid:
    - Care Takers always accept the job immediately if possible.

Normal forms

Our database schema is in neither BCNF nor 3NF.

The following are the non-trivial functional dependencies in our database.

- category_type -> base_price
- username -> password, uname, phone_number, area, position, credit_card
- ctuname, leave_date -> reason
- ctuname, category_type -> price
- pouname, pet_name -> category_type
- ctuname, pet_name, start_date, end_date, pouname -> has_paid, daily_price, is_completed, review, rating, transfer_method, payment_mode

The key is { username, leave_date, availability_date, pet_name, start_date, end_date, rtype, requirement }.

Justifications tables that are not in BCNF or 3NF of the whole schema are provided below.

- `category`:
    - `type` is not a superkey. So `category` is not in BCNF.
    - `base_price` is not a prime attribute. So `category` is not in 3NF.

- `users`:
  - `username` is not a superkey. So `users` is not in BCNF.
  - `password, uname, phone_number, area, position, credit_card` are not prime attributes. So `users` is not in 3NF.
- `apply_leave:`
  - (`username, date`) is not a superkey. So `apply_leave` is not in BCNF.
  - `reason` is not a prime attribute. So `apply_leave` is not in 3NF.
- `can_care`:
  - (`username, type`) is not a superkey. So `can_care` is not in BCNF.
  - `price` is not a prime attribute. So `can_care` is not in 3NF.
- `own_pet_belong`:
  - (`username, name`) is not a superkey. So `own_pet_belong` is not in BCNF.
- `take_care`:
  - (`ctuname, pet_name, start_date, end_date, pouname`) is not a superkey. So `take_care` is not in BCNF.
  - `has_paid, daily_price, is_completed, review, rating, transfer_method, payment_mode` are not prime attributes. So `take_care` is not in 3NF.

Nevertheless, our database has a BCNF decomposition:

- R1 (category_type, base_price)
- R2 (username, password, name, phone_number, area, position, credit_card)
- R3 (username, leave_date, reason, availability_date)
- R4 (username, category_type, price)
- R5 (username, pet_name, category_type, rtype, requirement)
- R6 (ctuname, pet_name, start_date, end_date, pouname, has_paid, daily_price, is_completed, review, rating, transfer_method, payment_mode)

The left hand side of each non-trivial functional dependency in each fragment is a superkey. This is enforced by primary key constraints in the database. Since all fragments (tables) are in BCNF, our database has a BCNF decomposition.

Interesting triggers

1. Add leave

Only full time Care Takers should be able to apply leave. Leave cannot be applied if leave is already applied on the date specified or if there is a job on the date specified. In addition, leave cannot be applied if the Care Taker is unable to work for a minimum of 2 x 150 consecutive days for the year.

```
CREATE OR REPLACE FUNCTION check_leave()
   RETURNS TRIGGER AS
   $$ DECLARE yr INTEGER;
   DECLARE ctx_a1 INTEGER;
   DECLARE ctx_a2 INTEGER;
   DECLARE prev DATE;
   DECLARE i INTEGER;
   DECLARE d DATE;
   DECLARE ctx INTEGER := 0;
```

```
   BEGIN
       IF NEW.username IN (SELECT * FROM part_time) THEN
           RAISE EXCEPTION 'Only full timer need to apply for leave!';
       ELSIF NEW.date IN (SELECT date FROM apply_leave a WHERE a.username =
NEW.username) THEN
           RAISE EXCEPTION 'Leave already applied on this date!';
       ELSIF 1 = (SELECT 1 FROM take_care t WHERE NEW.username = t.ctuname AND
(NEW.date >= t.start_date AND NEW.date <= t.end_date)) THEN
           RAISE EXCEPTION 'There is a job on this date!';
       ELSE
           yr := EXTRACT(year FROM NEW.date);

           /* find number of leave before NEW.date */
           SELECT COUNT(*) INTO ctx_a1
           FROM apply_leave a
           WHERE NEW.username = a.username
               AND EXTRACT(year FROM a.date) = yr
               AND NEW.date - a.date > 0;

           /* find number of leave after NEW.date */
           SELECT COUNT(*) INTO ctx_a2
           FROM apply_leave a
           WHERE NEW.username = a.username
               AND EXTRACT(year FROM a.date) = yr
               AND NEW.date - a.date < 0;

           prev := DATE(yr::TEXT || '-01-01');
           i := 0;
           LOOP
               EXIT WHEN i = ctx_a1;
               d := (SELECT a.date FROM apply_leave a WHERE NEW.username = a.username
AND EXTRACT(year FROM a.date) = yr AND NEW.date - a.date > 0 ORDER BY a.date ASC
LIMIT 1 OFFSET i);
               IF (d - prev - 1) >= 300 THEN
                   ctx := ctx + 2;
               ELSIF (d - prev - 1) >= 150 THEN
                   ctx := ctx + 1;
               END IF;
               prev := d;
               i := i + 1;
           END LOOP;
           IF (NEW.date - prev - 1) >= 300 THEN
               ctx := ctx + 2;
           ELSIF (NEW.date - prev - 1) >= 150 THEN
               ctx := ctx + 1;
           END IF;
```

```
            prev := NEW.date;
            i := 0;
            LOOP
                EXIT WHEN i = ctx_a2;
                d := (SELECT a.date FROM apply_leave a WHERE NEW.username = a.username
AND EXTRACT(year FROM a.date) = yr AND NEW.date - a.date < 0 ORDER BY a.date ASC
LIMIT 1 OFFSET i);
                IF (d - prev - 1) >= 300 THEN
                    ctx := ctx + 2;
                ELSIF (d - prev - 1) >= 150 THEN
                    ctx := ctx + 1;
                END IF;
                prev := d;
                i := i + 1;
            END LOOP;
            IF (DATE(yr::TEXT || '-12-31') - prev - 1) >= 300 THEN
                ctx := ctx + 2;
            ELSIF (DATE(yr::TEXT || '-12-31') - prev - 1) >= 150 THEN
                ctx := ctx + 1;
            END IF;

            IF ctx < 2 THEN
                RAISE EXCEPTION 'Cannot Apply for leave! 2 * 150 consecutive working
days in a year not met!';
            ELSE
                RETURN NEW;
            END IF;
        END IF;
    END;
    $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS add_leave ON apply_leave;
CREATE TRIGGER add_leave
BEFORE INSERT ON apply_leave
FOR EACH ROW EXECUTE PROCEDURE check_leave();
```

2. Add bid

There should be at most one bid involving the same Pet Owner and pet at any point in time. A Care Taker must not take care of pets that they cannot care for. A Care Taker cannot take care of pets when he is on leave. A full time Care Taker can take care of up to 5 pets at any point in time. A part time Care Taker can take care of up to 2 pets unless he has a good rating (4 out of 5), but no more than 5.

```
CREATE OR REPLACE FUNCTION check_bid()
    RETURNS TRIGGER AS
    $$ DECLARE ctx INTEGER;
    DECLARE cat INTEGER;
    DECLARE ok INTEGER;
```

```
    DECLARE rating INTEGER;
    BEGIN
        SELECT COUNT(*) INTO ctx
        FROM take_care t
        WHERE t.username = NEW.username
            AND t.name = NEW.name
            AND ((t.start_date >= NEW.start_date AND t.start_date <= NEW.end_date)
            OR (t.end_date >= NEW.start_date AND t.end_date <= NEW.end_date));

        IF ctx > 0 THEN
            RAISE EXCEPTION 'Clash with existing orders!';
        ELSE
            SELECT COUNT(*) INTO cat
            FROM own_pet_belong o
            WHERE o.username = NEW.username
                AND o.name = NEW.name
                AND o.type IN (SELECT c.type
                               FROM can_care c
                               WHERE c.username = NEW.ctuname);

            IF cat = 0 THEN
                RAISE EXCEPTION 'Sorry... Care taker cannot care for pet type. Please
choose another caretaker.';
            ELSE
                SELECT COUNT(*) INTO ok
                FROM take_care t
                WHERE t.ctuname = NEW.ctuname
                    AND ((t.start_date >= NEW.start_date AND t.start_date <=
NEW.end_date)
                        OR (t.end_date >= NEW.start_date AND t.end_date <=
NEW.end_date));
                IF (NEW.ctuname IN (SELECT * FROM full_time)) THEN
                    IF 1 = (SELECT 1 FROM apply_leave a WHERE a.username = NEW.ctuname
AND (a.date >= NEW.start_date AND a.date <= NEW.end_date)) THEN
                        RAISE EXCEPTION 'Caretaker is on leave.';
                    END IF;
                    IF ok < 5 THEN
                        RETURN NEW;
                    ELSE
                        RAISE EXCEPTION 'Sorry... Care taker not available. Please
choose another caretaker.';
                    END IF;
                ELSE
                    SELECT AVG(t.rating) INTO rating
                    FROM take_care t
                    WHERE t.ctuname = NEW.ctuname;
                    IF (rating IS NULL OR rating < 4) AND ok < 2 THEN
                        RETURN NEW;
```

```
                    ELSIF rating >= 4 AND ok < 5 THEN
                        RETURN NEW;
                    ELSE
                        RAISE EXCEPTION 'Sorry... Care taker not available. Please
choose another caretaker.';
                    END IF;
                END IF;
            END IF;
        END IF;
    END;
    $$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS add_bid ON take_care;
CREATE TRIGGER add_bid
BEFORE INSERT ON take_care
FOR EACH ROW EXECUTE PROCEDURE check_bid();
```

3. Add price

A Care Taker can decide on the daily price for each pet type that he can care for if he has a good rating (4 out of 5), but it should never fall below the base price specified by PCS. If the Care Taker does not have any rating or if the rating is less than 4 out of 5, he cannot set the daily price.

```
CREATE OR REPLACE FUNCTION check_price()
    RETURNS TRIGGER AS
    $$ DECLARE baseprice INTEGER;
    DECLARE rating INTEGER;
    BEGIN
        SELECT AVG(t.rating) INTO rating
        FROM take_care t
        WHERE t.ctuname = NEW.username;

        SELECT c.base_price INTO baseprice
        FROM category c
        WHERE c.type = NEW.type;

        IF rating >= 4 AND (NEW.price IS NULL OR NEW.price >= baseprice) THEN
            RETURN NEW;
        ELSIF (rating < 4 OR rating IS NULL) AND NEW.price IS NULL THEN
            RETURN NEW;
        ELSIF (rating < 4 OR rating IS NULL) THEN
            RAISE EXCEPTION 'Your rating is too low!';
        ELSE
            RAISE EXCEPTION 'Price should be higher than base price $%!', baseprice;
        END IF;
    END;
    $$ LANGUAGE plpgsql;
```

```
DROP TRIGGER IF EXISTS add_price on can_care;
CREATE TRIGGER add_price
BEFORE INSERT OR UPDATE ON can_care
FOR EACH ROW EXECUTE PROCEDURE check_price();
```

Complex queries

1. Get monthly summary for past 12 months

Returns the month, total number of transactions, total number of pets, total number of pet days, total salary, total earnings and total profit for all Care Takers for each month in the past 12 months.

```
SELECT to_char(to_timestamp (r.month::TEXT, 'MM'), 'Mon') AS month, r.year,
SUM(r.transactions) AS total_transactions,
   SUM(r.pets) AS total_pets, SUM(r.pet_days) AS total_pet_days,
   CAST(SUM(r.salary) AS DECIMAL(100,2)) AS total_salary, CAST(SUM(r.earnings) AS
DECIMAL(100,2)) AS total_earnings,
   CAST((SUM(r.earnings) - SUM(r.salary)) AS DECIMAL(100,2)) AS profit
FROM
   (SELECT f.username, 'Full Time' AS job_type, COUNT(*) AS transactions,
COUNT(DISTINCT t.name) AS pets,
       SUM(t.end_date - t.start_date + 1) AS pet_days, 3000 AS salary,
       SUM(t.daily_price * (t.end_date - t.start_date + 1)) AS earnings,
       t.emonth AS month, t.eyear AS year
   FROM full_time f INNER JOIN
       (SELECT *, EXTRACT(month FROM end_date) as emonth, EXTRACT(year FROM end_date)
AS eyear FROM take_care) AS t
       ON f.username = t.ctuname
   WHERE t.has_paid AND t.is_completed
   GROUP BY (f.username, t.emonth, t.eyear)
   HAVING COUNT(*) <= 60
   UNION
   SELECT f.username, 'Full Time' AS job_type, COUNT(*) AS transactions,
COUNT(DISTINCT t.name) AS pets,
       SUM(t.end_date - t.start_date + 1) AS pet_days,
       (3000 + (SELECT SUM(0.8 * daily_price * (end_date - start_date + 1))
               FROM take_care t1
               WHERE t1.ctuname = f.username
                   AND has_paid AND is_completed
                   AND EXTRACT(month FROM t1.end_date) = t.emonth
                   AND EXTRACT(year FROM t1.end_date) = t.eyear
               GROUP BY (t1.start_date, t1.end_date)
               ORDER BY t1.end_date ASC, t1.start_date ASC
               OFFSET 60)) AS salary,
       SUM(t.daily_price * (t.end_date - t.start_date + 1)) AS earnings,
       t.emonth AS month, t.eyear AS year
   FROM full_time f INNER JOIN
```

```
       (SELECT *, EXTRACT(month FROM end_date) as emonth, EXTRACT(year FROM end_date)
AS eyear FROM take_care) AS t
       ON f.username = t.ctuname
   WHERE t.has_paid AND t.is_completed
   GROUP BY (f.username, t.emonth, t.eyear)
   HAVING COUNT(*) > 60
   UNION
   SELECT p.username, 'Part Time' AS job_type, COUNT(*) AS transactions,
COUNT(DISTINCT t.name) AS pets,
       SUM(t.end_date - t.start_date + 1) AS pet_days,
       (SELECT SUM(0.75 * daily_price * (end_date - start_date + 1))
       FROM take_care t1
       WHERE t1.ctuname = p.username
           AND has_paid AND is_completed
           AND EXTRACT(month FROM t1.end_date) = t.emonth
           AND EXTRACT(year FROM t1.end_date) = t.eyear) AS salary,
       SUM(t.daily_price * (t.end_date - t.start_date + 1)) AS earnings,
       t.emonth AS month, t.eyear AS year
   FROM part_time p INNER JOIN
       (SELECT *, EXTRACT(month FROM end_date) as emonth, EXTRACT(year FROM end_date)
AS eyear FROM take_care) AS t
       ON p.username = t.ctuname
   WHERE t.has_paid AND t.is_completed
   GROUP BY (p.username, t.emonth, t.eyear)) AS r
WHERE (r.year = EXTRACT(year FROM now()) AND r.month < EXTRACT(month FROM now()))
   OR (r.year = EXTRACT(year FROM now()) - 1 AND r.month >= EXTRACT(month FROM
now()))
GROUP BY (r.month, r.year)
ORDER BY (r.year, r.month) ASC;
```

2. Get monthly Care Taker report

Returns the username, job type, total number of transactions, total number of pets, total number of pet days, total salary and total earnings for each Care Taker in the specified month and year.

```
SELECT f.username, 'Full Time' AS job_type, COUNT(*) AS transactions, COUNT(DISTINCT
t.name) AS pets,
   SUM(t.end_date - t.start_date + 1) AS pet_days, 3000.00 AS salary,
   CAST(SUM(t.daily_price * (t.end_date - t.start_date + 1)) AS DECIMAL(100,2)) AS
earnings,
   t.emonth AS month, t.eyear AS year
FROM full_time f INNER JOIN
   (SELECT *, EXTRACT(month FROM end_date) as emonth, EXTRACT(year FROM end_date) AS
eyear FROM take_care) AS t
   ON f.username = t.ctuname
WHERE t.emonth = ${month} AND t.eyear = ${year} AND t.has_paid AND t.is_completed
GROUP BY (f.username, t.emonth, t.eyear)
HAVING COUNT(*) <= 60
```

```
UNION
SELECT f.username, 'Full Time' AS job_type, COUNT(*) AS transactions, COUNT(DISTINCT
t.name) AS pets,
   SUM(t.end_date - t.start_date + 1) AS pet_days,
   CAST((3000 + (SELECT SUM(0.8 * daily_price * (end_date - start_date + 1))
               FROM take_care t1
               WHERE t1.ctuname = f.username
                   AND has_paid
                   AND is_completed
                   AND EXTRACT(month FROM t1.end_date) = t.emonth
                   AND EXTRACT(year FROM t1.end_date) = t.eyear
               GROUP BY (t1.start_date, t1.end_date)
               ORDER BY t1.end_date ASC, t1.start_date ASC
               OFFSET 60)) AS DECIMAL(100,2)) AS salary,
   CAST(SUM(t.daily_price * (t.end_date - t.start_date + 1)) AS DECIMAL(100,2)) AS
earnings,
   t.emonth AS month, t.eyear AS year
FROM full_time f INNER JOIN
   (SELECT *, EXTRACT(month FROM end_date) as emonth, EXTRACT(year FROM end_date) AS
eyear FROM take_care) AS t
   ON f.username = t.ctuname
WHERE t.emonth = ${month} AND t.eyear = ${year} AND t.has_paid AND t.is_completed
GROUP BY (f.username, t.emonth, t.eyear)
HAVING COUNT(*) > 60
UNION
SELECT p.username, 'Part Time' AS job_type, COUNT(*) AS transactions, COUNT(DISTINCT
t.name) AS pets,
   SUM(t.end_date - t.start_date + 1) AS pet_days,
   CAST((SELECT SUM(0.75 * daily_price * (end_date - start_date + 1))
           FROM take_care t1
           WHERE t1.ctuname = p.username
               AND has_paid
               AND is_completed
               AND EXTRACT(month FROM t1.end_date) = t.emonth
               AND EXTRACT(year FROM t1.end_date) = t.eyear) AS DECIMAL(100,2)) AS
salary,
   CAST(SUM(t.daily_price * (t.end_date - t.start_date + 1)) AS DECIMAL(100,2)) AS
earnings,
   t.emonth AS month, t.eyear AS year
FROM part_time p INNER JOIN
   (SELECT *, EXTRACT(month FROM end_date) as emonth, EXTRACT(year FROM end_date) AS
eyear FROM take_care) AS t
   ON p.username = t.ctuname
WHERE t.emonth = ${month} AND t.eyear = ${year} AND t.has_paid AND t.is_completed
GROUP BY (p.username, t.emonth, t.eyear);
```

3. Find Care Taker

Returns the username, name, phone number, area, average rating, daily price and job type of all Care Takers who can care for the pet type specified and are available during the period (start_date and end_date) specified.

```
SELECT u.username, u.name, u.phone_number, u.area, ROUND(ratings.rating, 2) as
rating,
   CAST(COALESCE(c.price,cat.base_price) AS DECIMAL(100,2)) as price, 'Full Time' as
job_type
FROM full_time f NATURAL JOIN users u NATURAL JOIN can_care c NATURAL JOIN category
cat
   LEFT JOIN (SELECT t.ctuname AS username, AVG(t.rating) AS rating FROM take_care t
GROUP BY t.ctuname) AS ratings ON f.username = ratings.username
WHERE c.type = '${type}'
   AND NOT EXISTS (SELECT 1
                FROM (SELECT date FROM apply_leave WHERE username = u.username) as d
                WHERE d.date >= DATE('${start_date}') AND d.date <=
DATE('${end_date}')) AND DATE('${start_date}') <= DATE('${end_date}')
   AND (SELECT COUNT(*)
       FROM take_care t1
       WHERE t1.ctuname = u.username
           AND ((t1.start_date >= DATE('${start_date}') AND t1.start_date <=
DATE('${end_date}'))
               OR (t1.end_date >= DATE('${start_date}') AND t1.end_date <=
DATE('${end_date}')))) < 5
UNION
SELECT u.username, u.name, u.phone_number, u.area, ROUND(ratings.rating, 2) as
rating,
   CAST(COALESCE(c.price,cat.base_price) AS DECIMAL(100,2)) as price, 'Part Time' as
job_type
FROM part_time p NATURAL JOIN users u NATURAL JOIN can_care c NATURAL JOIN category
cat
   LEFT JOIN (SELECT t.ctuname AS username, AVG(t.rating) AS rating FROM take_care t
GROUP BY t.ctuname) AS ratings ON p.username = ratings.username
WHERE c.type = '${type}'
   AND (SELECT COUNT(*)
       FROM specify_availability s
       WHERE s.username = p.username AND (date >= DATE('${start_date}') AND date <=
DATE('${end_date}')))
       = (DATE('${end_date}') - DATE('${start_date}')) + 1
   AND (((ratings.rating IS NULL OR ratings.rating < 4)
       AND (SELECT COUNT(*)
           FROM take_care t1
           WHERE t1.ctuname = u.username
               AND ((t1.start_date >= DATE('${start_date}') AND t1.start_date <=
DATE('${end_date}'))
                   OR (t1.end_date >= DATE('${start_date}') AND t1.end_date <=
DATE('${end_date}')))) < 2)
```

```
        OR (ratings.rating >= 4
            AND (SELECT COUNT(*)
                FROM take_care t1
                WHERE t1.ctuname = u.username
                    AND ((t1.start_date >= DATE('${start_date}') AND t1.start_date <=
DATE('${end_date}'))
                        OR (t1.end_date >= DATE('${start_date}') AND t1.end_date <=
DATE('${end_date}')))) < 5));
```

Software tools and frameworks

- Database: PostgreSQL
- Backend: NodeJS, Express, EJS
- Frontend: HTML, CSS, Bootstrap, JS

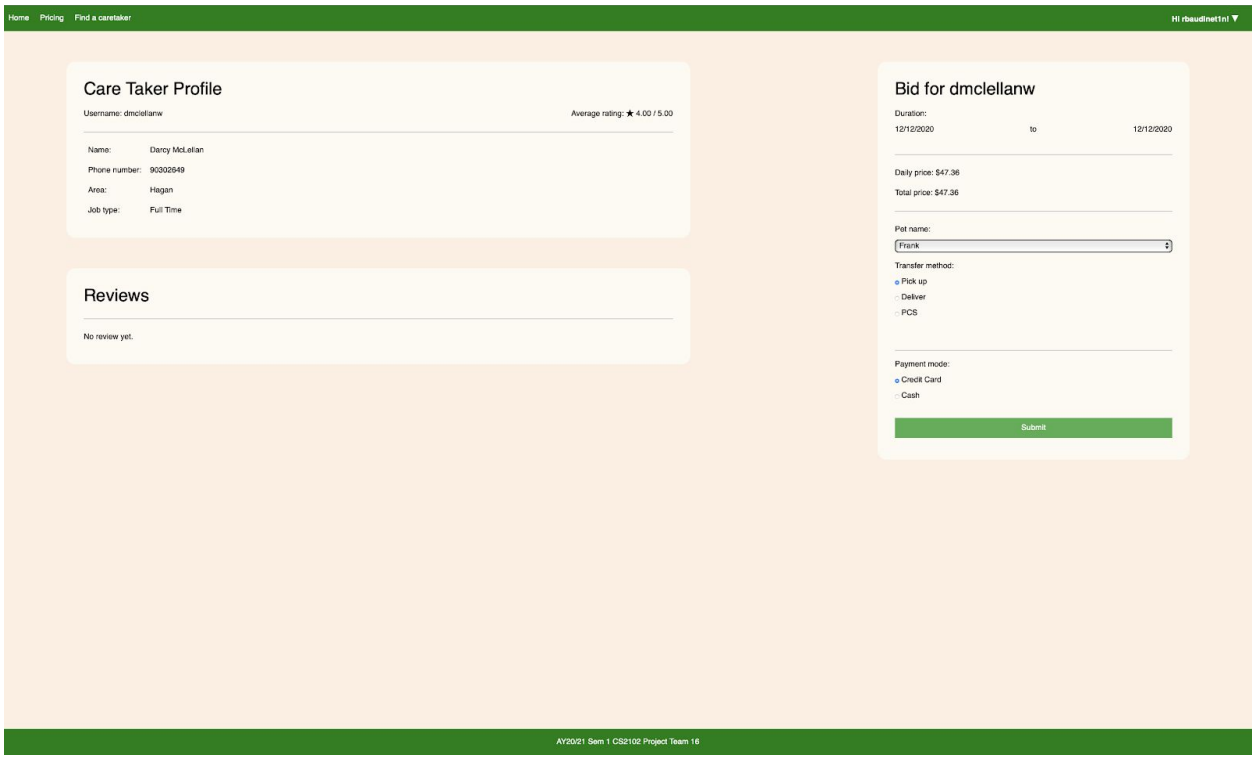
Screenshots

1. Profile page for care taker

Care takers can change the details of their account by clicking on the “edit profile” button which will redirect them to an edit profile page. They can also add the type of pet they would like to care for. If they are part-timers, they can declare their availability, if they are part-timers, they can apply for leaves. They can also choose to delete their profile from this page.

2. Bidding for a care taker

When a pet owner has chosen the service of a care taker that they would like to bid for, after clicking the bid button, the details of the care taker will be shown. They can then choose the transfer method and payment mode.



3. Admin summary

PCS administrators can view useful summary statistics, such as total salary, earnings and profit, and the monthly summary of all transactions within the past 12 months.

Difficulties and lessons learned

Being new to web programming, we faced several difficulties coding the web frontend and backend. We were unfamiliar with JavaScript and HTML. Due to our lack of experience, we were also unsure of the tools available and were faced with bugs when we tried new things. We spent a lot of time googling and debugging.

We faced challenges when we tried to populate the database. This is mostly because of the different table constraints such as foreign key constraints and data constraints that we want to enforce. We initially inserted into the table using data generators and manually checked for those records that violate constraints. However, as the amount of records to insert is large, it is not practical to do so. Thus, we used DO which is a transient anonymous function in PostgreSQL to help us populate and it was much easier. Triggers were also used to check that we enforced some of our data constraints, thus we realised that triggers are really useful.

We have learnt that it is important to have a good database design. A good database design helps to make the implementation of queries simpler and its execution faster. It also reduces data redundancy and allows easy maintenance. However, it is hard to change the database design halfway, thus we learnt that it is important to carefully plan our database before starting to code the website.

We had a great time working on the project and have learnt many new things throughout the project. We are satisfied with the end product although there is still room for improvement.