CS2102 Database Systems

Group 17 Project Report

Chen Yongyan	A0177871N
Han Yuxuan	A0187356R
Wang Ziyun	A0187304E
Yuan Jiayi	A0177893H
Zhang Yiping	A0192214M

7th November 2020

Table of Contents

1. Introduction 3

1.1 Project Requirements 3

1.2 Project Responsibilities for Each Member 3

2. Description of Website 3

2.1 Functionalities 3

2.1.1 User information 3

2.1.2 Regulation of Caretakers 4

2.1.3 Pet Caring Service 4

2.1.4 Administrative Activities 5

2.2 Constraints 5

3. Entity-Relationship (ER) Model 9

4. Schema 9

4.1 Relational Schema 9

4.2 Database Normalization 12

5. Implementation Highlights 13

5.1 Non-trivial / Interesting Triggers 13

5.1.1 Minimum Working Days Requirement 13

5.1.2 Increasing Salary with every Successful Bid 14

5.1.3 Updating Remaining Limit and Daily Price 15

5.2 Most Complex Queries 16

5.2.1 Searching for Caretaker 16

5.2.2 Caretaker’s Monthly Statistics 17

5.2.3 Underperforming Full Time Caretakers 18

6. Software Tools / Frameworks Used 19

7. Difficulties and Lesson Learned 19

1. Introduction

1.1 Project Requirements

For this project, our team is tasked to develop a database application for Pet Caring Service (PCS). The application will support three kinds of **users**: Caretakers, Pet Owners and PCS Administrators (Admin). Our pet owners can also register as caretakers and vice versa. **Caretakers** are employees for the PCS company, they will be **capable** of taking care of certain **categories** of pets. **Pet owners** can search and **bid** for caretakers who are **available** for work to take care of their **pets**. From every successful bid, caretakers will obtain **salaries** paid by the admins. More details of functionalities of our website will be illustrated in section 2.1 of the report.

1.2 Project Responsibilities for Each Member

Project Member	**Repsonsibilities**
Chen Yongyan	ER Diagram, Schema, Triggers, Backend Web Development, Report
Han Yuxuan	Constraints, Schema, Triggers, Backend Web Development
Wang Ziyun	Constraints, Schema, Triggers, Queries, Report
Yuan Jiayi	ER Diagram, Schema, Frontend Web Development
Zhang Yiping	ER Diagram, Schema, Frontend Web Development

2. Description of Website

2.1 Functionalities

This section provides brief descriptions for functionalities supported by our website. More data constraints and implementation details can be found under section 2.2.

2.1.1 User information

Our website supports browsing of information for caretakers and pet owners. Common information includes phone number, name, transfer location for transfering of the pets. Meanwhile, pet owners also have information for their pets, which includes pets' names, the categories of the pets (e.g., Dogs, Cats, etc.) and special requirements on how the pets should be taken care of, if any. To facilitate interactions between pet owners and caretakers, caretakers have to specify the categories of pets they are capable of and the time during which they are available for work. Their average rating of past services will also be available to the pet owners. Using this information, pet owners can then look for caretakers and bid for their pet caring service.

2.1.2 Regulation of Caretakers

Caretakers at the PCS company are classified into full time and part time and the company has different policies applied to them to regulate the quality of their services.

- ❏ Care limit sets the number of pets per day a caretaker can service. All full time caretakers have a care limit of 5, but part time caretakers have a care limit of 2 if their average ratings are below 4, and 5 otherwise.
- ❏ Part time caretakers can set their daily price of servicing each category of pets on our website, but full time caretakers' daily prices follow the base prices set by the admins if their average rating is below 4 and increase by (average rating - 4) * 10 otherwise.
- ❏ Part time caretakers can specify their available working days, while full time caretakers are assumed to be available all the time and can apply for leave. Full time caretakers have to work for 2 * 150 consecutive days per year and the system will automatically accept any bid that they are available for and capable of.
- ❏ Full time caretakers have a base monthly salary of $3000. Full time caretakers also have a feature named pet-day that reflects their monthly workload (1 pet-day per pet per day serviced). Part time caretakers' monthly income is 75% of the cost of their services in the month. Full time caretakers' monthly salary will not increase before their pet-day reaches 60 and obtain 80% of the cost of their services afterwards.

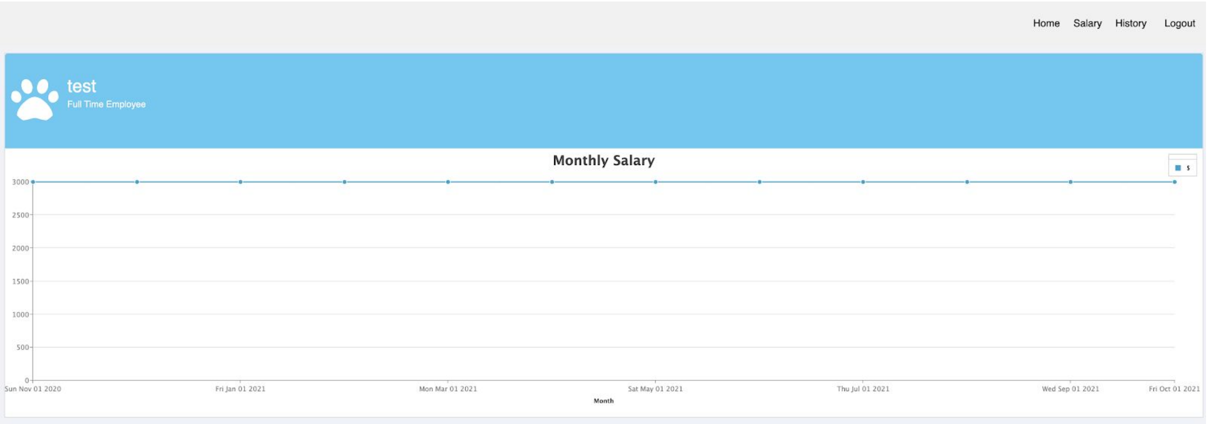
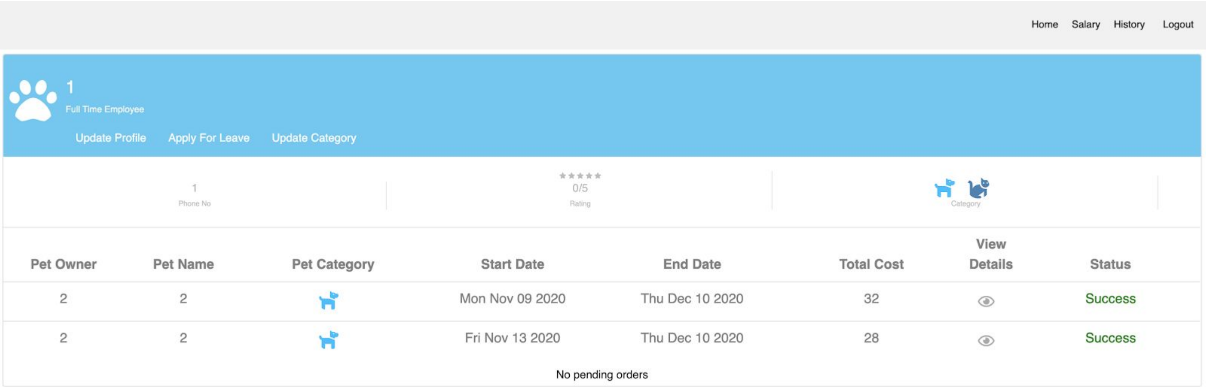


Figure 1: Monthly Salary Page for a Full Time Caretaker

2.1.3 Pet Caring Service

To set sufficient time for both caretakers and pet owners to respond to a bid, we set some time constraints for both parties. We ask all pet owners to bid for a service at least 3 days in advance and all pet owners to accept a bid at least 2 day in advance. All pet caring services have to be paid upfront either by cash or by credit card, hence the pet owners will have to pay for an accepted bid at least 1 day in advance. Once the monetary transaction is complete, both parties cannot change their minds and the service fee will not be refunded. The pet can be transferred either by pet owner delivery, caretaker pick-up or via the physical building of PCS. Both the caretaker and pet owner should contact each other privately to agree on the transfer method. The pet owners can rate and comment after the end of each successful bid.



Pet Owner	Pet Name	Pet Category	Start Date	End Date	Total Cost	View Details	Status
2	2	🐕	Mon Nov 09 2020	Thu Dec 10 2020	32	👁	Success
2	2	🐕	Fri Nov 13 2020	Thu Dec 10 2020	28	👁	Success

No pending orders

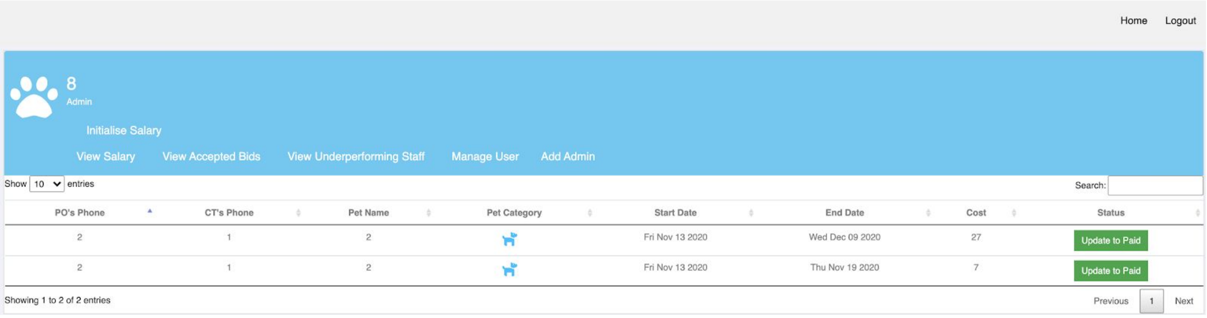
Figure 2: Caretaker View Successful Bids Page

2.1.4 Administrative Activities

The website supports various functionalities for PCS admins. Primarily, only existing admins can register new admins into the database, except for the very first admin directly initialized at the backend.

For regulation of users, admins are able to delete existing pet owners and caretakers from the database. Admins can initialize default salary for full time caretakers on a yearly basis. By clicking ‘View Salary’, admins will see all salaries that have not been paid to the caretakers. We also allow admins to view underperforming full time caretakers whose monthly pet-day does not exceed average pet-day for 9 months in a year.

As shown in Figure 3, admins will be able to view all accepted bids and change their status to ‘Success’ after receiving service fees paid by cash.



PO's Phone	CT's Phone	Pet Name	Pet Category	Start Date	End Date	Cost	Status
2	1	2	🐕	Fri Nov 13 2020	Wed Dec 09 2020	27	Update to Paid
2	1	2	🐕	Fri Nov 13 2020	Thu Nov 19 2020	7	Update to Paid

Showing 1 to 2 of 2 entries

Figure 3: Admins View Accepted Bids Page

2.2 Constraints

The data constraints of our application are broken down into entities, relationships and aggregates in accordance with our Entity-Relationship(ER) Diagram in section 3.1. Majority of these constraints are realized by triggers in PostgreSQL on the back-end while others are enforced on the front-end of our website.

1. Users (Entity)
   1.1. Users are uniquely identified by their phone number.

   1.2. A user can be a pet owner, a caretaker, both a pet owner and a caretaker or an admin.
   1.3. For each user, name and password must be recorded.
2. Pet Owners (Entity)
   2.1. Every pet owner is a user.
   2.2. For each pet owner, a credit card number and transfer location may or may not be recorded.
   2.3. Each pet owner must own at least one pet.
3. Caretakers (Entity)
   3.1. Every caretaker is a user.
   3.2. Care limit (maximum number of pets that can be taken care of in a day) and a bank account must be recorded for each caretaker.
   3.3. For each caretaker, transfer location may or may not be recorded.
   3.4. Caretakers are either part time caretakers or full time caretakers but cannot be both.
   3.5. Average ratings for past successful bids received by the caretaker, ranging from 0 to 5, must be recorded. If the caretaker has yet to receive any rating, the default average rating is 0.
   3.6. For each full time caretaker, the care limit is 5.
   3.7. For each part time caretaker, the care limit is 2 if the average rating is below 4, otherwise the care limit is 5.
4. Pets (Weak Entity of Pet Owners)
   4.1. Pets are identified by the pet owner’s phone number and the pet name.
   4.2. Special requirements may or may not be recorded.
   4.3. Category of the pet must be recorded, and the category must be one of the categories recorded under the category table.
   4.4. Every pet can be owned by exactly one pet owner.
   4.5. A pet is deleted if its owner is deleted.
5. Category (Entity)
   5.1. Category is uniquely identified by category name.
   5.2. Base daily price must be recorded for each category.
6. Capable (Relationship)
   6.1. For every caretaker, he\she must be capable of taking care of at least one category of pets.
   6.2. For each capability under every caretaker, there must be a daily price recorded.
   6.3. For the full-time caretakers, the daily price must not be lower than the base daily price of the category.
   6.4. For full-time caretakers with average rating above 4, the daily price is equal to base price + (average rating - 4) * 10.
   6.5. For part-time caretakers, they can set the daily price however they want.
7. Availability (weak entity of Caretakers)
   7.1. Availability is identified by the caretaker’s phone number and the available date on which the caretaker is working.
   7.2. The remaining limit (number of pets the caretaker can still take care of on that date) must be recorded for each available date of each caretaker. Once the caretaker is employed for a new successful bid, the remaining limit on that date is updated accordingly.

7.3. An entry in availability is deleted if the caretaker is deleted.

7.4. For each full time caretaker, he\she must be available for 2 * 150 consecutive days a year (from 1 Jan to 31 Dec).

7.5. For each full time caretaker, he\she is assumed to be available everyday for the current and next year.

7.6. The default availability for full time caretakers is initialized either automatically at the point of registration or manually by admins at the start of a year.

7.7. Full time caretakers can apply leave on dates that they are not taking care of any pets (remaining limit equals care limit) and the corresponding dates will be removed from available.

7.8. If there are pending / accepted bids on the day which the caretaker wants to take leave, the system will automatically reject the bids.

7.9. For each part time caretaker, he\she can specify the dates that they are available for the current and next year.

8. Bids (Aggregate)

8.1. A pet owner can place a bid for service for a pet from a caretaker for at least 3 days in advance. The 3 days leave sufficient time for caretakers to respond to the bid.

8.2. A bid is uniquely identified by the pet owner’s phone, the caretaker’s phone, the pet name, the start and end date.

8.3. Details of a bid including status, category of the pet, daily price, total price, transfer method and payment method must be recorded.

8.4. The status of a bid includes ‘Pending’, ‘Withdraw’, ‘Accepted’, ‘Rejected’, ‘Success’ and ‘Fail’.

8.5. A bid that is freshly placed by a pet owner has the status ‘Pending’. A pet owner can withdraw a bid before the monetary transaction is completed. The caretaker who receives a bid can either reject or accept the bid for at least 2 days in advance. The 2 days leave sufficient time for pet owners to pay for the accepted bid upfront. The system will automatically accept a bid for an available full-time caretaker.

8.6. Monetary transactions can only be made after a bid is accepted by the caretaker and must be made a day in advance. If a pet owner fails to pay for an accepted bid before its start date, the bid will be marked as ‘Fail’, otherwise ‘Success’.

8.7. A pet owner with a credit card can be automatically charged for an accepted bid via the card. For accepted bids paid by cash, the admin will change the status to ‘Success’.

8.8. Pet owners can only rate and comment after the end of successful bids, the rate is an integer between 1 to 5 and the comment can be anything within 500 characters.

8.9. Transfer method of each bid can either be (1) pet owner deliver, (2) caretaker pick up or (3) transfer through the physical building of PCS. For caretakers and pet owners who do not wish to record their locations, they will have to private message each other for transfering of the pet.

8.10. A pet can only be taken care of by a caretaker if all of the following are satisfied: (1) the caretaker is available (2) the number of pets he/she is taking care of on that day has not meet his/her care limit (3) the category of the pet matches the caretaker’s capability.

8.11. The total cost is calculated by duration * daily price.

8.12. The payment method can either be by cash or by credit card if the pet owner has a pre-registered credit card.

9. Admin (Entity)

9.1. Every admin is a user.

9.2. Only existing admins can register new admins into the database, except for the very first admin initialized directly at the backend.

10. Pay (Relationship)

10.1. All payments can be uniquely identified by the caretaker's phone number and pay time. The phone number of the admin who was in charge of the payment is also recorded.

10.2. A recorded payment means the corresponding salary has been paid to the caretaker by the admin.

11. Salary (Entity)

11.1. Salary is identified by the caretaker's phone number and pay time (first day of every month), and the amount paid must be recorded. If the caretaker is a full time, pet-day must be recorded.

11.2. Salary is deleted if the corresponding caretaker is deleted.

11.3. The default salary amount is 3000 for a full time caretaker and 0 for a part time caretaker. The default pet-day is 0 for a full time caretaker at the start of each month.

11.4. For a full-time caretaker, the default salary is initialized for every month in the current and next year either automatically at the point of registration or manually by admins at the start of a year.

11.5. For a part-time caretaker, the default salary for a month is initialized automatically before the first successful bid of the month.

12. Increase (Relationship)

12.1. Salary increases with every successful bid.

12.2. If the caretaker is part time, the amount of salary for that month increases by 75% of the total cost of the bid.

12.3. If the caretaker is full time, while his\her pet-day < 60 for the month, for each day of servicing each pet, the bid will increase his\her pet-day by 1; after his\her pet-day reaches 60, for each successful bid, the amount of salary increases by 80% of the total cost of service (3000 + bonus).

3. Entity-Relationship (ER) Model

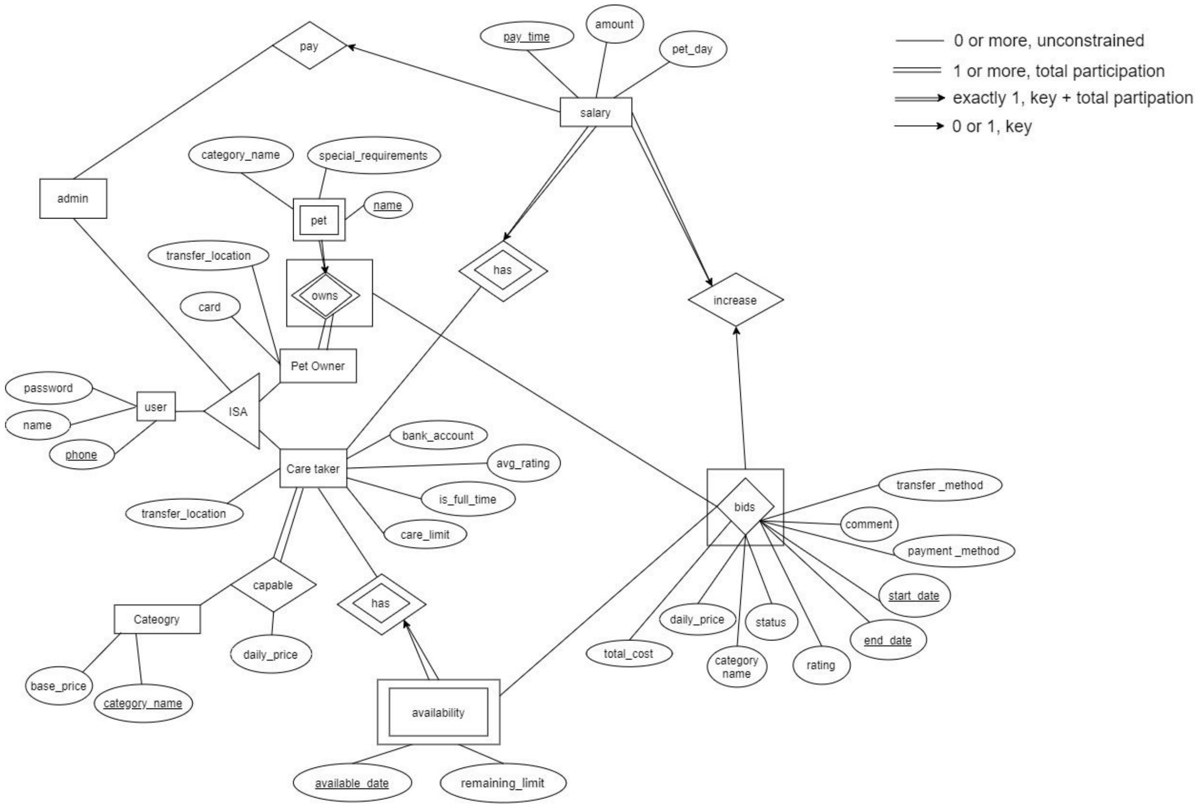


Figure 4: Entity-Relationship Diagram

As shown in Figure 4, the use of an ER Diagram provides a clear visualization of our database design, effectively establishing relationships between multiple tables and capturing key data constraints. The list of constraints from section 2.2 that are not captured by our ER model includes point 1.2, 3.5-3.7, 4.3, 6.3-6.5, 7.4-7.9, 8.4-8.12, 9.2, 11.3-11.5 and 12.2-12.3.

4. Schema

4.1 Relational Schema

The relational schema derived from our ER model is as displayed below as PostgreSQL CREATE TABLE statements. Some of the constraints mentioned in section 2.2 are enforced by column or table constraints. The list of constraints that are not enforced includes point 2.3, 6.1, 6.3-6.4, 7.4-7.8, 8.1, 8.5-8.7, 8.10, 9.2, 11.3-11.5 and 12.1-12.3.

```
CREATE TABLE users(
      phone INTEGER PRIMARY KEY,
      password VARCHAR NOT NULL,
      role VARCHAR NOT NULL
      CHECK(role IN ('Pet Owner', 'Caretaker', 'Both', 'Admin')),
      UNIQUE(phone, password)
);
```

```
CREATE TABLE pet_owner(
      phone INTEGER,
      password VARCHAR NOT NULL,
      transfer_location VARCHAR,
      name VARCHAR NOT NULL,
      card VARCHAR(16),
      PRIMARY KEY(phone),
      FOREIGN KEY (phone, password) REFERENCES users(phone, password)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE care_taker(
      phone INTEGER,
      password VARCHAR NOT NULL,
      transfer_location VARCHAR,
      name VARCHAR NOT NULL,
      bank_account VARCHAR NOT NULL,
      is_full_time BOOLEAN NOT NULL,
      avg_rating FLOAT8 DEFAULT 0
      CHECK(avg_rating >= 0 AND avg_rating <= 5),
      care_limit INTEGER NOT NULL
      CHECK(care_limit = CASE
            WHEN is_full_time IS TRUE THEN 5
            WHEN is_full_time IS NOT TRUE AND avg_rating < 4 THEN 2
            WHEN is_full_time IS NOT TRUE AND avg_rating >= 4 THEN 5
            END),
      PRIMARY KEY (phone),
      FOREIGN KEY (phone, password) REFERENCES users(phone, password)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE admin(
      phone INTEGER PRIMARY KEY,
      password VARCHAR NOT NULL,
      name VARCHAR NOT NULL,
      FOREIGN KEY (phone, password) REFERENCES users(phone, password)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE category(
      category_name VARCHAR PRIMARY KEY,
      base_price FLOAT8 NOT NULL
);

CREATE TABLE owns_pet(
      phone INTEGER NOT NULL REFERENCES pet_owner(phone)
      ON DELETE CASCADE ON UPDATE CASCADE,
      name VARCHAR,
      special_requirements VARCHAR(500),
      category_name VARCHAR NOT NULL REFERENCES category(category_name),
      PRIMARY KEY(phone, name),
      UNIQUE(phone, name, category_name)
```

```
);

CREATE TABLE availability(
      phone INTEGER REFERENCES care_taker(phone)
      ON DELETE CASCADE ON UPDATE CASCADE,
      available_date DATE,
      remaining_limit INTEGER CHECK(remaining_limit >= 0),
      PRIMARY KEY (phone, available_date),
      UNIQUE(phone, available_date)
);

CREATE TABLE capable(
      phone INTEGER REFERENCES care_taker(phone)
      ON DELETE CASCADE ON UPDATE CASCADE,
      category_name VARCHAR REFERENCES category(category_name),
      daily_price FLOAT8 NOT NULL,
      PRIMARY KEY(phone, category_name),
      UNIQUE(phone, category_name, daily_price)
);

CREATE TABLE salary(
      phone INTEGER REFERENCES care_taker(phone)
      ON DELETE CASCADE ON UPDATE CASCADE,
      pay_time DATE,
      amount FLOAT8 DEFAULT 0,
      pet_day INTEGER,
      PRIMARY KEY(phone, pay_time)
);

CREATE TABLE pay(
      ad_phone INTEGER NOT NULL REFERENCES admin(phone)
      ON DELETE CASCADE ON UPDATE CASCADE,
      ct_phone INTEGER,
      pay_time DATE,
      PRIMARY KEY(ct_phone, pay_time),
      FOREIGN KEY (ct_phone, pay_time) REFERENCES salary(phone, pay_time)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE bids(
      po_phone INTEGER,
      ct_phone INTEGER,
      pet_name VARCHAR,
      start_date DATE,
      end_date DATE CHECK(end_date >= start_date),
      status VARCHAR NOT NULL DEFAULT 'Pending' CHECK(status IN (
            'Pending', 'Withdraw', 'Rejected',
            'Accepted', 'Success', 'Fail')),
      category_name VARCHAR NOT NULL,
      daily_price FLOAT8 NOT NULL,
      transfer_method VARCHAR NOT NULL
      CHECK(transfer_method IN ('PO deliver', 'CT pick up', 'via PCS')),
      total_cost FLOAT8 NOT NULL
      CHECK (total_cost = daily_price * (end_date - start_date + 1)),
      payment_method VARCHAR NOT NULL
      CHECK(payment_method IN ('Cash', 'Credit Card')),
      rating INTEGER CHECK(rating >= 1 and rating <= 5),
      comment VARCHAR(500),
      PRIMARY KEY (po_phone, ct_phone, pet_name, start_date, end_date),
      FOREIGN KEY (po_phone, pet_name, category_name)
      REFERENCES owns_pet(phone, name, category_name)
      ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (ct_phone, category_name)
      REFERENCES capable(phone, category_name)
      ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (ct_phone, start_date)
      REFERENCES availability(phone, available_date)
      ON DELETE CASCADE ON UPDATE CASCADE,
      FOREIGN KEY (ct_phone, end_date)
      REFERENCES availability(phone, available_date)
      ON DELETE CASCADE ON UPDATE CASCADE
);
```

4.2 Database Normalization

Set of schema consists of the following fragments:

- `users{phone -> password, role}`
- `admin{phone -> name, password}`
- `pet_owner{phone -> card, transfer_location, name, password}`
- `care_taker{phone -> is_full_time, avg_rating, care_limit, transfer_location, name, bank_account, password}`
- `owns_pet{name, phone -> special_requirements, category_name}`
- `availability{phone, available_date -> remaining_limit}`
- `category{category_name -> base_price}`
- `capable{phone, category_name -> daily_price}`
- `salary{phone, pay_time -> amount, pet_day}`
- `pay{ct_phone, pay_time -> ad_phone}`
- `bids{po_phone, ct_phone, pet_name, start_date, end_date -> category_name, daily_price, transfer_method, total_cost, payment_method, rating, comment, status}`

In the relational schema, all attributes at the left-hand side are superkeys, and all the other attributes are allowed to have duplicates and cannot infer any of other attributes using functional dependencies. Hence, every attribute is directly dependent on all keys and none of them is transitively dependent on any key. As a result, our database satisfies **BCNF** properties.

5. Implementation Highlights

5.1 Non-trivial / Interesting Triggers

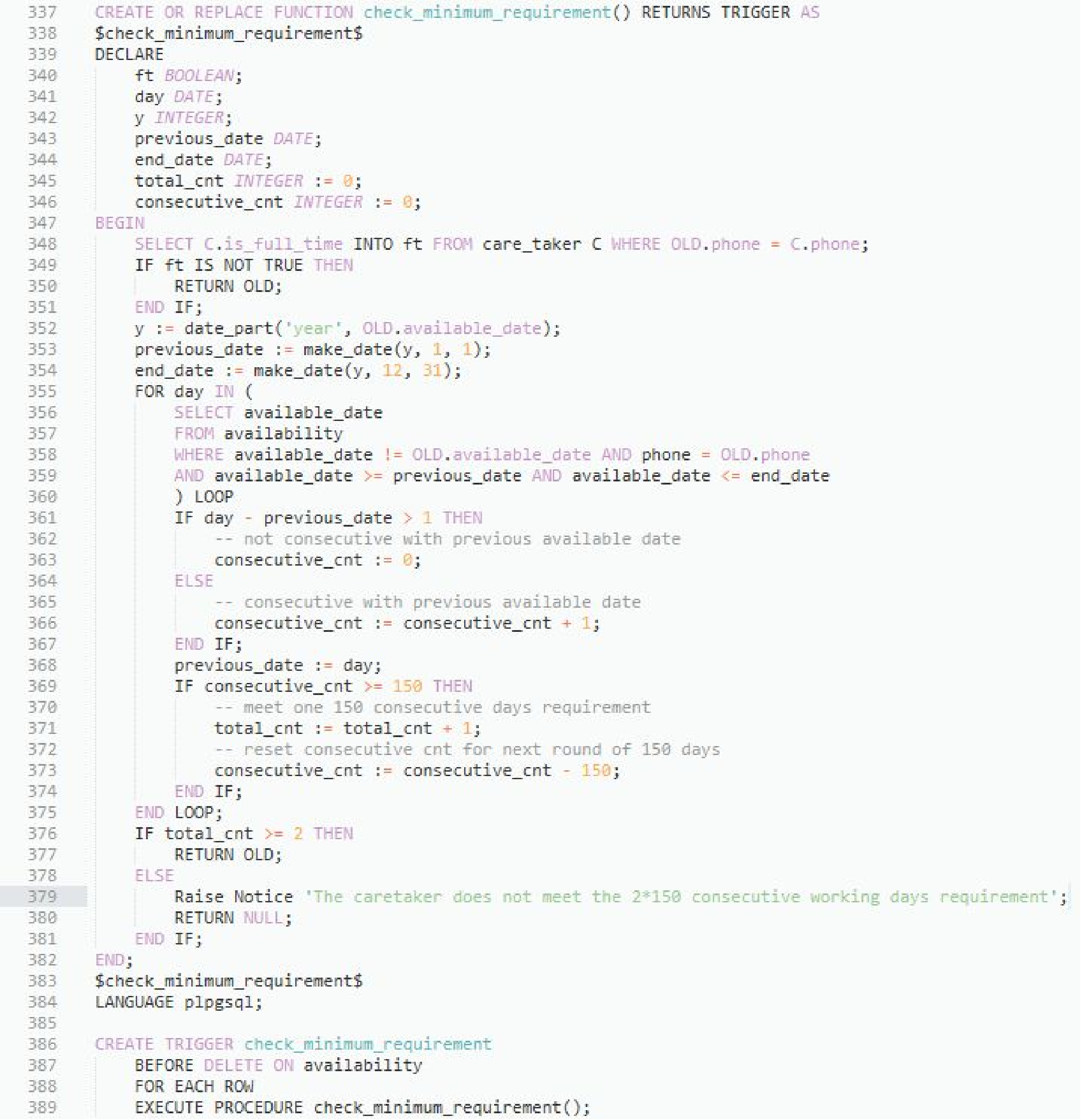
We are unable to present all our 16 triggers implemented for the database due to page constraints. Instead, we would like to illustrate the top three most complicated triggers of our database application in this section.

5.1.1 Minimum Working Days Requirement

To enforce section 2.2 point 7.4, we implemented a trigger named **check_minimum_requirement** which is activated before deletion of an entry from the availability table (take leave).

The trigger first checks whether it’s a part-time caretaker who’s taking leave and permits the deletion if it is. This is because only the full-time caretakers are subjected to the minimum working days requirement. If it’s a full-time caretaker, the trigger then checks if the deletion results in a violation of the minimum working days requirement and refuses the deletion if it does. This is done via a for loop through all remaining available dates after the deletion and counting consecutive days. If the remaining available dates are able to form 2 * 150 consecutive days, the deletion will not result in a violation of minimum working days requirement and will be permitted.

The PostgreSQL code for the trigger is as shown below.



```
CREATE OR REPLACE FUNCTION check_minimum_requirement() RETURNS TRIGGER AS
$check_minimum_requirement$
DECLARE
    ft BOOLEAN;
    day DATE;
    y INTEGER;
    previous_date DATE;
    end_date DATE;
    total_cnt INTEGER := 0;
    consecutive_cnt INTEGER := 0;
BEGIN
    SELECT C.is_full_time INTO ft FROM care_taker C WHERE OLD.phone = C.phone;
    IF ft IS NOT TRUE THEN
        RETURN OLD;
    END IF;
    y := date_part('year', OLD.available_date);
    previous_date := make_date(y, 1, 1);
    end_date := make_date(y, 12, 31);
    FOR day IN (
        SELECT available_date
        FROM availability
        WHERE available_date != OLD.available_date AND phone = OLD.phone
        AND available_date >= previous_date AND available_date <= end_date
        ) LOOP
        IF day - previous_date > 1 THEN
            -- not consecutive with previous available date
            consecutive_cnt := 0;
        ELSE
            -- consecutive with previous available date
            consecutive_cnt := consecutive_cnt + 1;
        END IF;
        previous_date := day;
        IF consecutive_cnt >= 150 THEN
            -- meet one 150 consecutive days requirement
            total_cnt := total_cnt + 1;
            -- reset consecutive cnt for next round of 150 days
            consecutive_cnt := consecutive_cnt - 150;
        END IF;
    END LOOP;
    IF total_cnt >= 2 THEN
        RETURN OLD;
    ELSE
        Raise Notice 'The caretaker does not meet the 2*150 consecutive working days requirement';
        RETURN NULL;
    END IF;
END;
$check_minimum_requirement$
LANGUAGE plpgsql;

CREATE TRIGGER check_minimum_requirement
    BEFORE DELETE ON availability
    FOR EACH ROW
    EXECUTE PROCEDURE check_minimum_requirement();
```

Figure 5: check_minimum_requirement Trigger

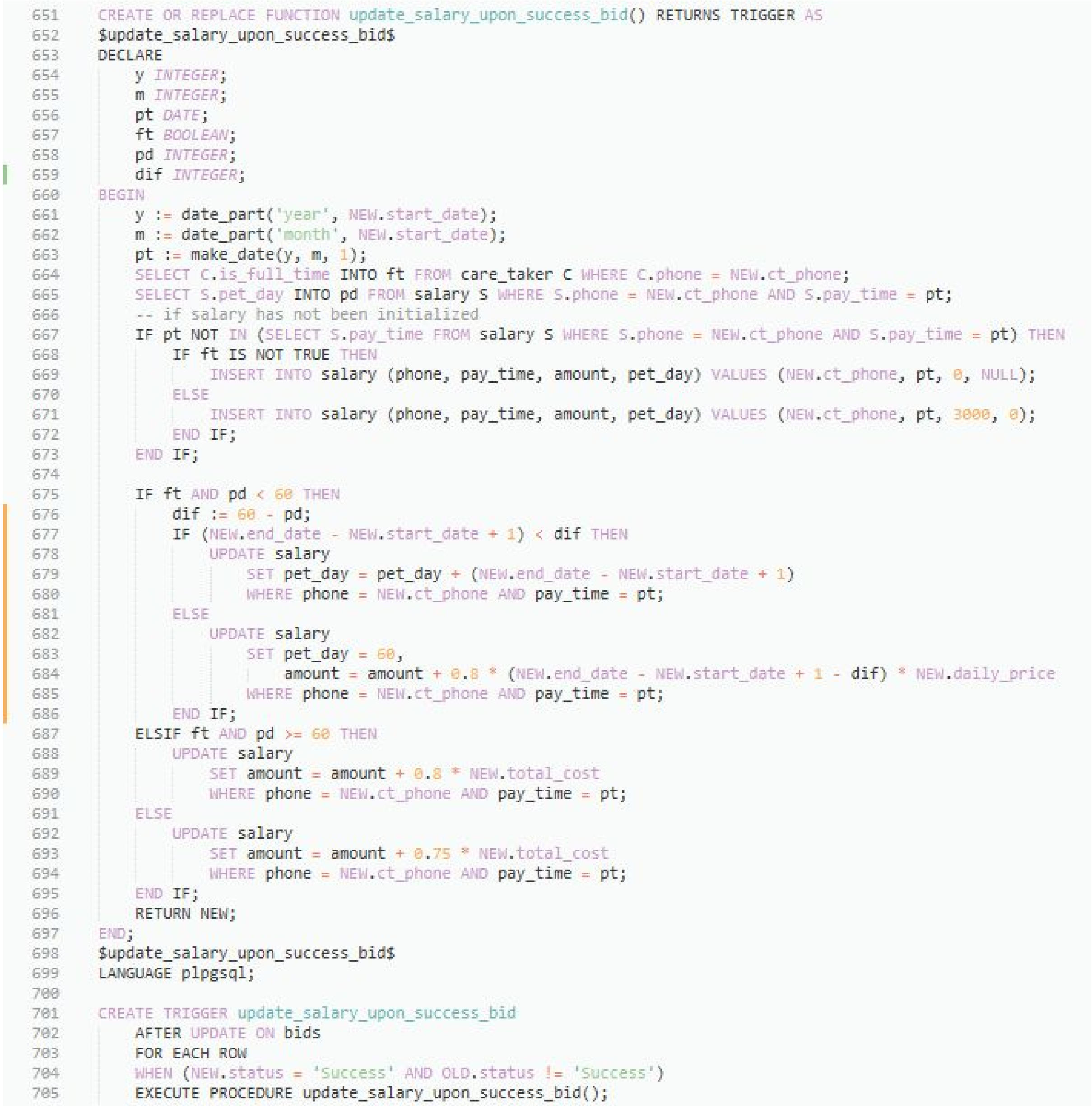
5.1.2 Increasing Salary with every Successful Bid

To enforce section 2.2 point 11.5, 12.1-12.3, we implemented a trigger named **update_salary_upon_success_bid** which is activated after every update on the bids table where the status of a bid is changed to 'Success'.

The trigger first checks if the default salary for the caretaker for the month has been initialized. If the answer is negative, the trigger will initialize the default salary for the caretaker for the month, following the rule at section 2.2 point 11.3.

The trigger then update the salary table according to the following rule: if the caretaker is full time, while his/her pet-day < 60, increase the pet-day by 1 for each day of the bid until it reaches 60, then increase the salary amount by 80% of the daily price for each exceeding day; if the caretaker is part time, increase the salary amount by 0.75 of the total cost.

The PostgreSQL code for the trigger is as shown below.



```
CREATE OR REPLACE FUNCTION update_salary_upon_success_bid() RETURNS TRIGGER AS
$update_salary_upon_success_bid$
DECLARE
    y INTEGER;
    m INTEGER;
    pt DATE;
    ft BOOLEAN;
    pd INTEGER;
    dif INTEGER;
BEGIN
    y := date_part('year', NEW.start_date);
    m := date_part('month', NEW.start_date);
    pt := make_date(y, m, 1);
    SELECT C.is_full_time INTO ft FROM care_taker C WHERE C.phone = NEW.ct_phone;
    SELECT S.pet_day INTO pd FROM salary S WHERE S.phone = NEW.ct_phone AND S.pay_time = pt;
    -- if salary has not been initialized
    IF pt NOT IN (SELECT S.pay_time FROM salary S WHERE S.phone = NEW.ct_phone AND S.pay_time = pt) THEN
        IF ft IS NOT TRUE THEN
            INSERT INTO salary (phone, pay_time, amount, pet_day) VALUES (NEW.ct_phone, pt, 0, NULL);
        ELSE
            INSERT INTO salary (phone, pay_time, amount, pet_day) VALUES (NEW.ct_phone, pt, 3000, 0);
        END IF;
    END IF;

    IF ft AND pd < 60 THEN
        dif := 60 - pd;
        IF (NEW.end_date - NEW.start_date + 1) < dif THEN
            UPDATE salary
                SET pet_day = pet_day + (NEW.end_date - NEW.start_date + 1)
                WHERE phone = NEW.ct_phone AND pay_time = pt;
        ELSE
            UPDATE salary
                SET pet_day = 60,
                    amount = amount + 0.8 * (NEW.end_date - NEW.start_date + 1 - dif) * NEW.daily_price
                WHERE phone = NEW.ct_phone AND pay_time = pt;
        END IF;
    ELSIF ft AND pd >= 60 THEN
        UPDATE salary
            SET amount = amount + 0.8 * NEW.total_cost
            WHERE phone = NEW.ct_phone AND pay_time = pt;
    ELSE
        UPDATE salary
            SET amount = amount + 0.75 * NEW.total_cost
            WHERE phone = NEW.ct_phone AND pay_time = pt;
    END IF;
    RETURN NEW;
END;
$update_salary_upon_success_bid$
LANGUAGE plpgsql;

CREATE TRIGGER update_salary_upon_success_bid
    AFTER UPDATE ON bids
    FOR EACH ROW
    WHEN (NEW.status = 'Success' AND OLD.status != 'Success')
    EXECUTE PROCEDURE update_salary_upon_success_bid();
```

Figure 6: update_salary_upon_success_bid Trigger

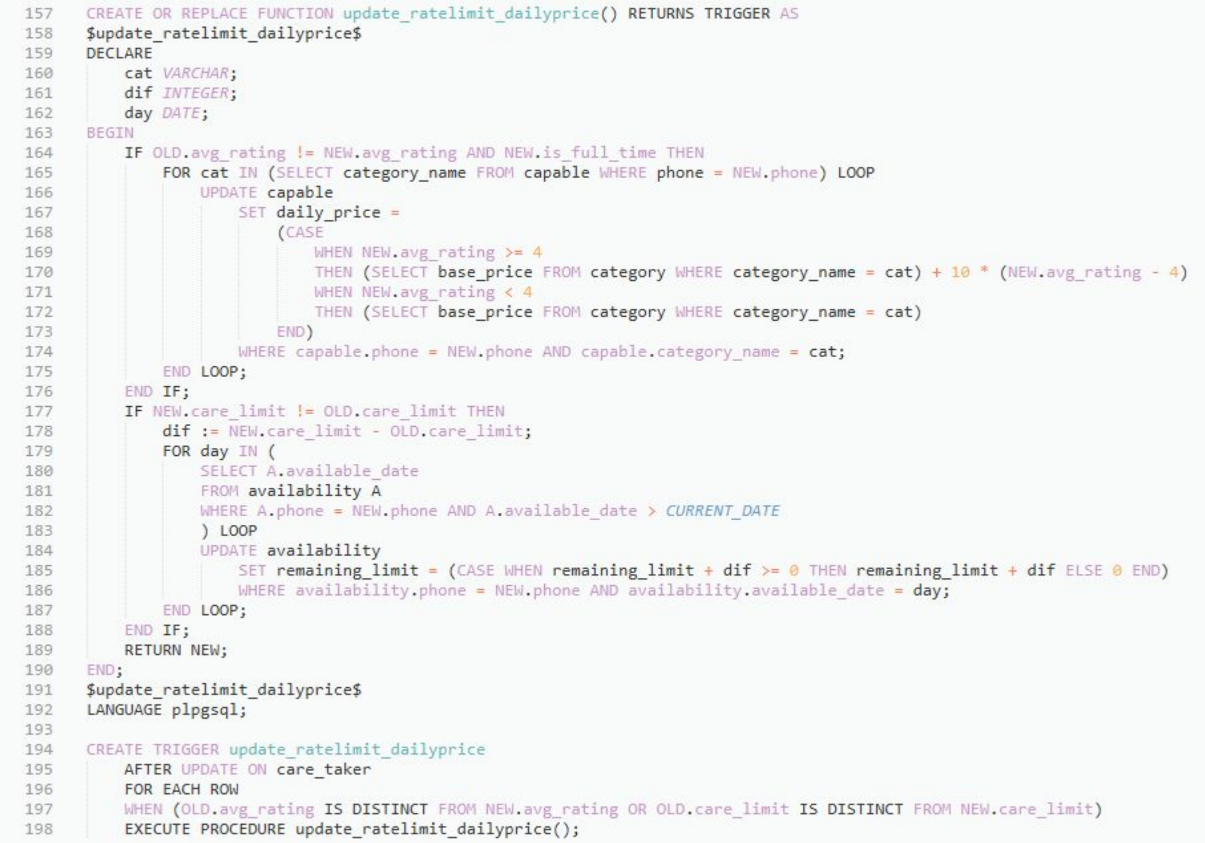
5.1.3 Updating Remaining Limit and Daily Price

To enforce section 2.2 point 6.3-6.4, 7.2, we implemented a trigger named **update_ratelimit_dailyprice** which is activated after every update on the care_taker table where the average rating or care limit of a caretaker is changed.

If the average rating for a caretaker has changed, the trigger will first check if the caretaker is full time. This is because constraints 6.3-6.4 only affects full time caretakers. If the caretaker is full time, then for every category of pets which the caretaker is capable of, the daily price is equivalent to base price + 10 * (average rating - 4) if average rating >= 4 and equivalent to base price if average rating < 4.

If the care limit of a caretaker has changed, for every available day of this caretaker in the future, the remaining limit is changed accordingly. If the care limit increases, the remaining limit will increase by the same amount; if the care limit decreases, the remaining limit will decrease by the same amount if the resulting remaining limit is non-negative, or reduced to 0 otherwise.

The PostgreSQL code for the trigger is as shown below.



```
CREATE OR REPLACE FUNCTION update_ratelimit_dailyprice() RETURNS TRIGGER AS
$update_ratelimit_dailyprice$
DECLARE
    cat VARCHAR;
    dif INTEGER;
    day DATE;
BEGIN
    IF OLD.avg_rating != NEW.avg_rating AND NEW.is_full_time THEN
        FOR cat IN (SELECT category_name FROM capable WHERE phone = NEW.phone) LOOP
            UPDATE capable
                SET daily_price =
                    (CASE
                        WHEN NEW.avg_rating >= 4
                        THEN (SELECT base_price FROM category WHERE category_name = cat) + 10 * (NEW.avg_rating - 4)
                        WHEN NEW.avg_rating < 4
                        THEN (SELECT base_price FROM category WHERE category_name = cat)
                    END)
                WHERE capable.phone = NEW.phone AND capable.category_name = cat;
        END LOOP;
    END IF;
    IF NEW.care_limit != OLD.care_limit THEN
        dif := NEW.care_limit - OLD.care_limit;
        FOR day IN (
            SELECT A.available_date
            FROM availability A
            WHERE A.phone = NEW.phone AND A.available_date > CURRENT_DATE
            ) LOOP
            UPDATE availability
                SET remaining_limit = (CASE WHEN remaining_limit + dif >= 0 THEN remaining_limit + dif ELSE 0 END)
                WHERE availability.phone = NEW.phone AND availability.available_date = day;
        END LOOP;
    END IF;
    RETURN NEW;
END;
$update_ratelimit_dailyprice$
LANGUAGE plpgsql;

CREATE TRIGGER update_ratelimit_dailyprice
    AFTER UPDATE ON care_taker
    FOR EACH ROW
    WHEN (OLD.avg_rating IS DISTINCT FROM NEW.avg_rating OR OLD.care_limit IS DISTINCT FROM NEW.care_limit)
    EXECUTE PROCEDURE update_ratelimit_dailyprice();
```

Figure 7: update_ratelimit_dailyprice Trigger

5.2 Most Complex Queries

Just as in section 5.1, for this section, we would like to present 3 of our most complex queries and illustrate how they work.

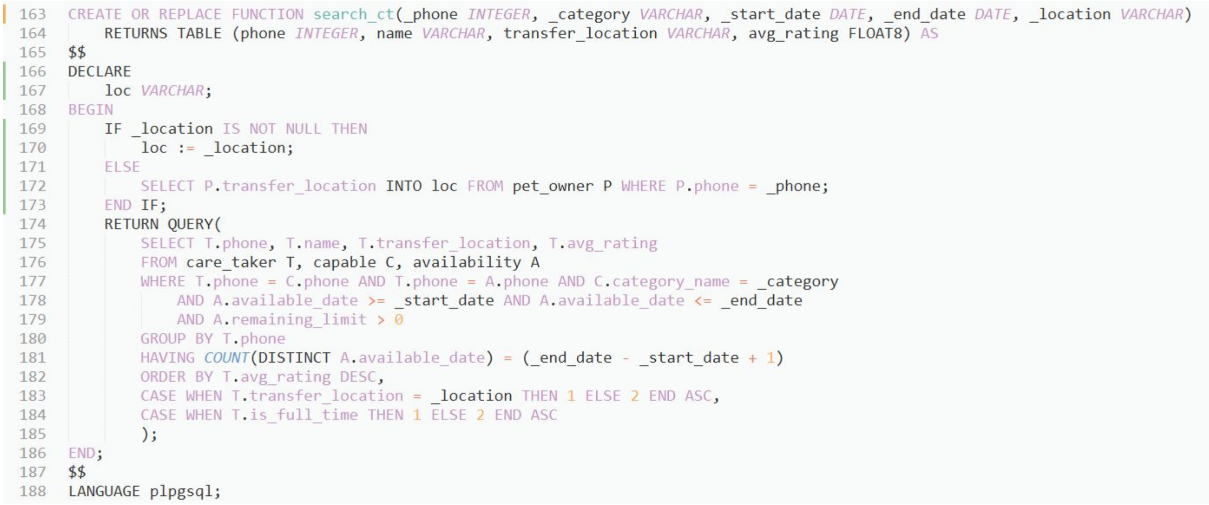
5.2.1 Searching for Caretaker

To allow pet owners to search for caretakers to place a particular bid, we implemented a function named **search_ct** which returns a set of queries to support this activity.

The function takes as input the pet owner’s phone number, the category of the pet involved in the bid, the start and end date of the bid and an optional transfer location. The optional here suggests that the input transfer location can be NULL. In the case that the input transfer location is NULL, the function will look for the registered transfer location of the pet owner in the pet_owner table.

The function returns a set of queries that looks for all the caretakers who are capable of the category and available with a positive remaining limit from start date to end date of this bid. It then orders them with the following rules: 1. descending with average rating of caretakers; 2. caretakers with the same transfer location will be placed higher; 3. full time caretakers will be placed higher. Finally, the queries return information about these caretakers as a table: caretaker's phone number, caretaker's name, caretaker's transfer location and caretaker's average rating.

The PostgreSQL code for the function is as shown below.



```
CREATE OR REPLACE FUNCTION search_ct(_phone INTEGER, _category VARCHAR, _start_date DATE, _end_date DATE, _location VARCHAR)
    RETURNS TABLE (phone INTEGER, name VARCHAR, transfer_location VARCHAR, avg_rating FLOAT8) AS
$$
DECLARE
    loc VARCHAR;
BEGIN
    IF _location IS NOT NULL THEN
        loc := _location;
    ELSE
        SELECT P.transfer_location INTO loc FROM pet_owner P WHERE P.phone = _phone;
    END IF;
    RETURN QUERY(
        SELECT T.phone, T.name, T.transfer_location, T.avg_rating
        FROM care_taker T, capable C, availability A
        WHERE T.phone = C.phone AND T.phone = A.phone AND C.category_name = _category
            AND A.available_date >= _start_date AND A.available_date <= _end_date
            AND A.remaining_limit > 0
        GROUP BY T.phone
        HAVING COUNT(DISTINCT A.available_date) = (_end_date - _start_date + 1)
        ORDER BY T.avg_rating DESC,
        CASE WHEN T.transfer_location = _location THEN 1 ELSE 2 END ASC,
        CASE WHEN T.is_full_time THEN 1 ELSE 2 END ASC
        );
END;
$$
LANGUAGE plpgsql;
```

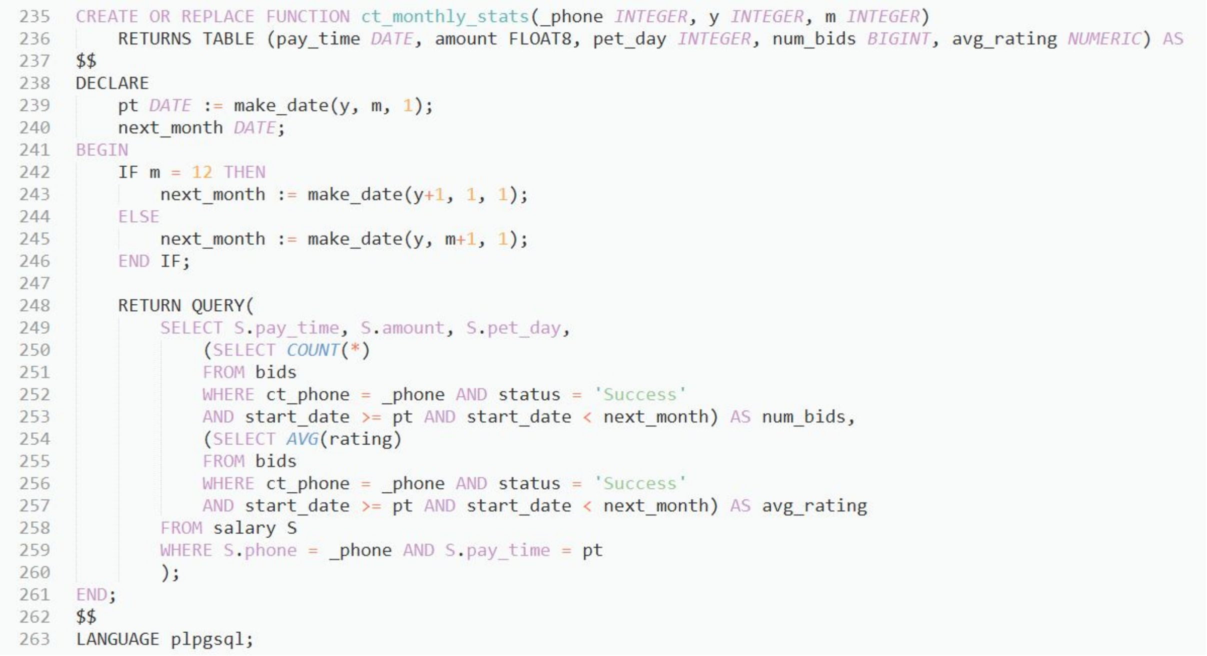
Figure 8: search_ct Function

5.2.2 Caretaker's Monthly Statistics

To allow caretakers or admins to view a caretaker's workload on a particular month, we implemented a function named **ct_monthly_stats** which returns a set of queries to support this activity.

The function takes as input the caretaker's phone number, the year and month number. It then looks for the caretaker's salary amount and pet-day for the specified month and also the number of successful bids and average rating for the month. Notice that the average rating here is different from the one in the care_taker table. The average rating here calculates the average rating in the specified month only, while the one in the care_taker table calculates the average rating of all time. Finally, the queries return information on the caretaker's workload for the specified month as a table: pay-time, salary amount, pet-day, number of successful bids and average rating.

The PostgreSQL code for the function is as shown below.



```sql
CREATE OR REPLACE FUNCTION ct_monthly_stats(_phone INTEGER, y INTEGER, m INTEGER)
    RETURNS TABLE (pay_time DATE, amount FLOAT8, pet_day INTEGER, num_bids BIGINT, avg_rating NUMERIC) AS
$$
DECLARE
    pt DATE := make_date(y, m, 1);
    next_month DATE;
BEGIN
    IF m = 12 THEN
        next_month := make_date(y+1, 1, 1);
    ELSE
        next_month := make_date(y, m+1, 1);
    END IF;

    RETURN QUERY(
        SELECT S.pay_time, S.amount, S.pet_day,
            (SELECT COUNT(*)
            FROM bids
            WHERE ct_phone = _phone AND status = 'Success'
            AND start_date >= pt AND start_date < next_month) AS num_bids,
            (SELECT AVG(rating)
            FROM bids
            WHERE ct_phone = _phone AND status = 'Success'
            AND start_date >= pt AND start_date < next_month) AS avg_rating
        FROM salary S
        WHERE S.phone = _phone AND S.pay_time = pt
        );
END;
$$
LANGUAGE plpgsql;
```

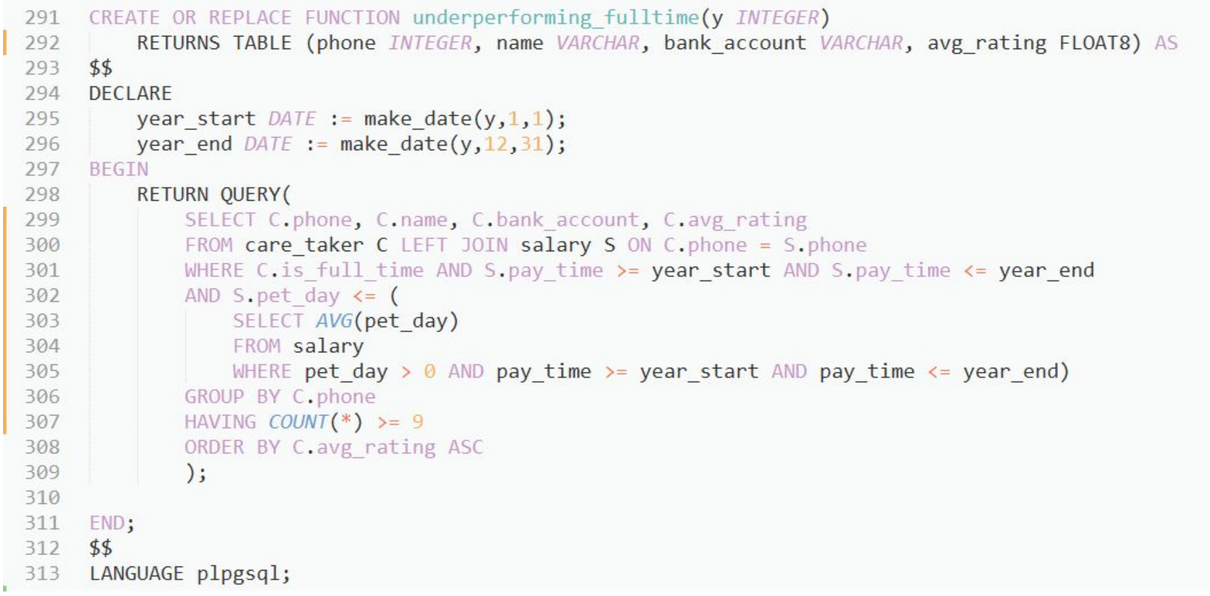
Figure 9: ct_monthly_stats Function

5.2.3 Underperforming Full Time Caretakers

To allow admins to view all underperforming full time caretakers, we implemented a function named **underperforming_fulltime** which returns a set of queries to support this activity.

This function takes as input a particular year. It then looks for all full time caretakers whose monthly pet-day is below the average pet-day for more than or equals to 3 quarters (9 months) in the year. Since pet-days and salary amounts are often automatically initialized at the start of the year with default pet-day 0, we will eliminate the 0 pet-days from the salary table when calculating the average pet-day to prevent large numbers of 0 pet-days from lowering the average. These underperforming full time caretakers are then ordered by ascending sequence of their average ratings, meaning the admins will see the underperforming full time caretaker with the lowest average rating first. Finally, the queries return information on these caretakers as a table: phone number, name, bank account and average rating.

The PostgreSQL code for the function is as shown below.



```
CREATE OR REPLACE FUNCTION underperforming_fulltime(y INTEGER)
    RETURNS TABLE (phone INTEGER, name VARCHAR, bank_account VARCHAR, avg_rating FLOAT8) AS
$$
DECLARE
    year_start DATE := make_date(y,1,1);
    year_end DATE := make_date(y,12,31);
BEGIN
    RETURN QUERY(
        SELECT C.phone, C.name, C.bank_account, C.avg_rating
        FROM care_taker C LEFT JOIN salary S ON C.phone = S.phone
        WHERE C.is_full_time AND S.pay_time >= year_start AND S.pay_time <= year_end
        AND S.pet_day <= (
            SELECT AVG(pet_day)
            FROM salary
            WHERE pet_day > 0 AND pay_time >= year_start AND pay_time <= year_end)
        GROUP BY C.phone
        HAVING COUNT(*) >= 9
        ORDER BY C.avg_rating ASC
        );

END;
$$
LANGUAGE plpgsql;
```

Figure 10: underperforming_fulltime Function

6. Software Tools / Frameworks Used

Below is a list of tools and packages that we used in development of our website:

- ❏ Node.js, Express
- ❏ PostgreSQL
- ❏ dotenv
- ❏ passport, bcrypt
- ❏ ejs, connect-flash, jQuery
- ❏ Bootstrap4
- ❏ npm, nodemon

7. Difficulties and Lesson Learned

Throughout this project, our team encountered many technical difficulties, primarily due to our lack of experience with database design and web development.

We did not have the perfect ER model and constraints designed before we started working on implementation in PostgreSQL, thus we found ourselves making changes in database designs and implementations back and forth. This wasted a lot of time especially when we had to keep the front and back end consistent. Our lesson learned is that, in the future, we will put in more time and efforts to design the database more carefully before diving into actual implementations.

Meanwhile, we, as students of CS2102, are newcomers to database systems and thus lack the experience coding in PostgreSQL. This project involved massive amounts of complicated triggers and queries that had to be implemented in PostgreSQL, which cost us a long time coding and debugging, mostly fixing syntax errors. Even though the lectures were helpful in giving sample codes, our experience with PostgreSQL can only improve with time and practice.

In addition, most of our team members have little to no experience with JavaScript, making web development particularly difficult for us. Even though there were guiding instructions given and self-sourced tutorial videos from YouTube, the learning process was still very painful, especially given the complexity of the requirements of this website. Yet again, we will gradually get better at this technique given time and practice.

In conclusion, most of our challenges evolved around the conflict between the high complexity of the project and our unfamiliarity with the technical aspects. We believe that having a solid database design and the necessary learning process with the technical tools is the key to efficiently work out a database application. Other than that, proper time management and consistent communication are also essential for a team project.