CS2102

Pet Pals - Team 58

Aakanksha Rai (A0184721E)

Abhiman Yadav (A0194091A)

Tan Wei Yang (A0173236H)

Abdulhusein Poonawala (A0177629M)

Joshua Wong (A0183871W)

Project Responsibilities

Name	Responsibilities
Aakanksha Rai	Wrote basic CRUD queries, salary related queries (salary calculation per caretaker, salary list for all caretakers, total expenses, revenue and profit, etc) and backend code for administrator related functionalities. Reviewed code.
Tan Wei Yang	Wrote a query for searching for matching caretakers. Wrote triggers for leave and availability constraints (2x150 consecutive day constraint, no overlap in dates). Added backend routes for caretaker search, and CRUD of caretaker services. Reviewed code.
Abdulhusein Jabir Poonawala	Wrote CRUD queries for pets, queries to get number of pets in specified month, get month with the highest number of jobs, see past reviews of pet owners and past jobs of caretakers. Wrote trigger to handle the addition of leaves for full_timers when they already have booking in place. Added most pet related routes for the backend. Reviewed code.
Abhiman Yadav	Wrote queries for fetching information about caretakers, trigger for updating caretaker bids upon update of prices, trigger for automatically declining clashing bids after caretaker accepts a bid and backend implementation of all routes related to registration, login and CRUD of booking. Created script to enter dummy data in database. Reviewed code.
Joshua Wong	Wrote the entire frontend app in React, fixed backend bugs, opened remaining CRUD routes and missing SQL queries, reviewed code.

Interesting Functionalities

- Bookings made for full-timers are automatically accepted if the full-timer hasn't met 60 pet days in any of the months that occur during the booking period and isn't over-booked for any time period during the booking.
- If the full-timer has already met 60 pet days for each of the relevant months, they have the choice to accept or decline the booking.
- Part-timers always have the choice to accept or decline their bookings.
- Whenever a booking is accepted, all other bookings that clash with it (i.e, bookings during periods that now have the caretaker booked to capacity due to the acceptance of this booking) will be automatically declined.

Constraints

Note: *Constraints not captured by the ER diagram are underlined.*

1. Users can be identified by their username, and must have a name, password and location.
2. Users must be at least one of the following roles: administrator, caretaker and pet owners and can be more than one of them (covering constraints must be met).
3. Users can register at most 1 credit card at a time.
4. Caretakers must be either Part-time or Full-time and can only be either one of these roles (Covering satisfied but not overlap).
5. All caretakes must have an average rating, equal to the average of ratings in their bookings. When the caretaker has not done any job yet (that is, they are new), they have a default average rating of 0.

6. Part-time caretakers can have multiple periods of availability, and are otherwise considered as unavailable.
7. Full-time caretakers can have multiple periods of leave, and are otherwise considered as available.
8. Each full-time caretaker must work for a minimum of 2x150 consecutive days a year.
9. Pets must have a name, type and can be associated with several services they requires
10. Pets are uniquely identified by their name and their owner's username. The name of pets uniquely identifies it among other pets owned by the same pet owner.
11. Pets cannot exist without an owner.
12. Each pet type is uniquely identified by the name of the animal, and is associated with a base price that is set by an admin.
13. For each pet type caretakers handle, they can charge their own price which must be greater than or equal to the base price of handling that pet type.
14. Full-time caretakers cannot apply for leave during periods where there are one or more pets under their care.
15. Full-time caretakers can only have a limit of up to 5 pets at any one time.
16. Full-time caretakers can apply for leaves only in the current year and the next year.
17. Part-time caretakers can specify their availability only in the current year and the next year.
18. Part time caretakers can only take care of up to 2 pets unless they have a rating of at least 4, where they can take care of up to 4 pets.
19. Bookings can only be created and accepted in time periods when the caretaker is available (see constraints 6 and 7), and if it does not violate the capacity of the caretaker (see constraints 16 and 20) at the given point of time. Pending and rejected bookings do not count towards these constraints.
20. Bookings with full-time caretakers are automatically accepted if the caretaker has not had 60 pet-days in that month.
21. Pending bookings are automatically rejected if the caretaker is not available, or has reached their capacity.
22. The bid price of the booking must be greater than the asking price of the caretaker for the respective pet type.
23. For each booking associated with pets owned by a pet owner, the pet owner may submit up to 1 review, including a rating from 0 to 5 and remarks.
    Note: Some information such as salary is not stored in our database, and the constraints of those fields are therefore not written here. They are calculated during the execution of the program
24. Leaves or availability periods of the same caretaker cannot overlap (for example, if there is a leave from 1st May 2020 to 6th May 2020, there cannot be a leave from 2nd May 2020 to 7th May 2020).

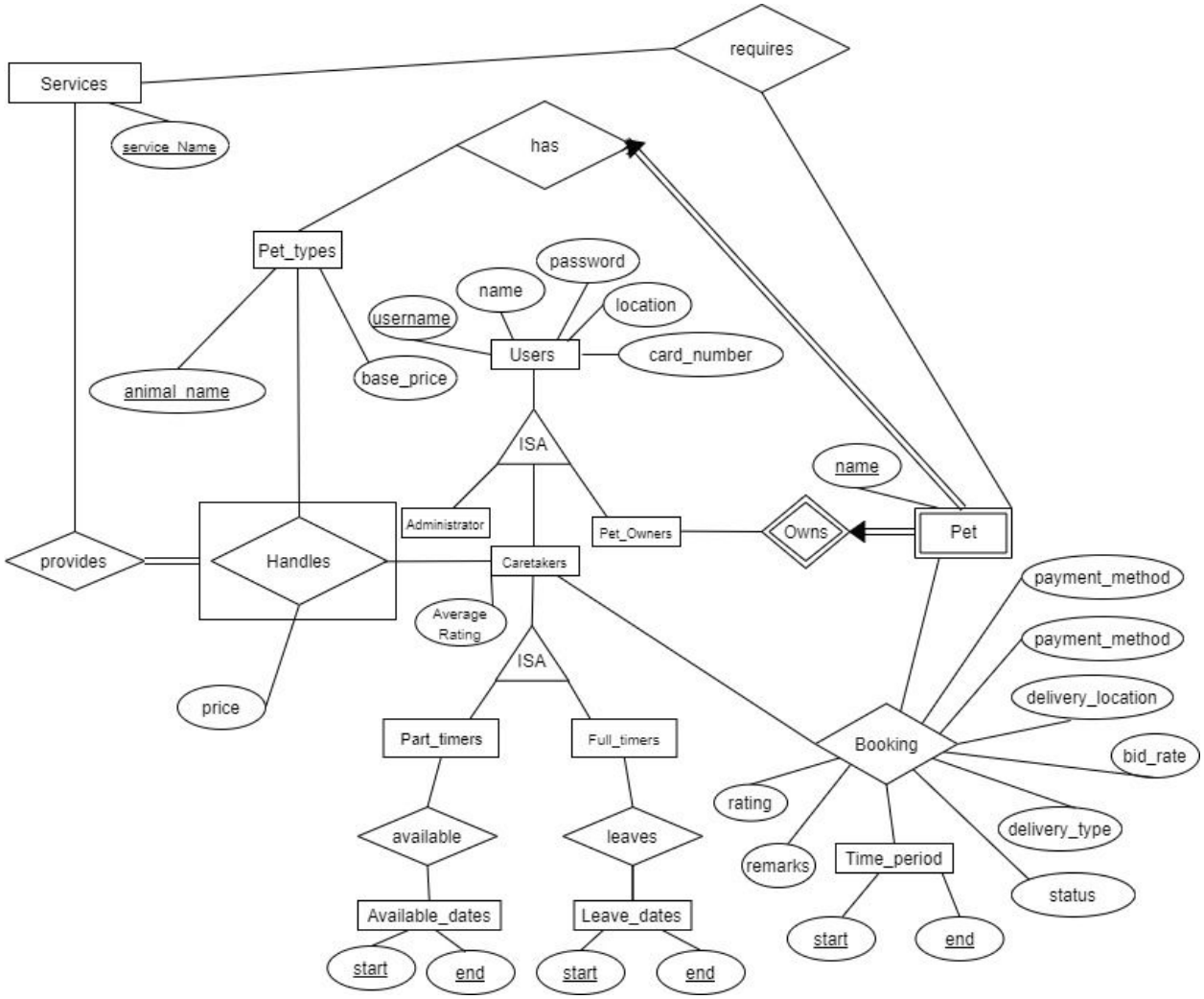
Non-Trivial Design Decisions

- Availability stores only the availability of part-time workers and not full time workers as part-time workers are only available on the specified dates and unavailable otherwise. The leaves table stores the leave dates of full-time workers as they are considered to be available unless on leave.
- Since the availability table doesn't contain details of full-time workers, we decided to associate each booking with a time period instead. We have triggers that make sure that this time period isn't during a full-timer's leave or is during a part-timer's availability (depending on whether the caretaker is a part-timer or a full-timer).
- We decided to have a services table and give it defined values so that it will be easier to match the services a caretaker provides to what a pet needs. This also ensures that a caretaker doesn't have to do

something they aren't prepared for. We have prepared the services table with every possible service that can be required/ is offered by our company.

- We have average rating as an attribute rather than a calculated field since we weren't allowed to use many CTEs for the project. The average rating is updated by a trigger whenever a new review is given.

ER Diagram



Relational Schema

```
CREATE TABLE users (
  username    varchar(64)  PRIMARY KEY,
  password    varchar(64)  NOT NULL,
  first_name  varchar(64)  NOT NULL,
   location    varchar(64)  NOT NULL,
   card_number varchar(16)  NOT NULL
);

CREATE TABLE administrator (
```

```
  username varchar(64) references users(username) PRIMARY KEY
);

CREATE TABLE caretakers (
  username varchar(64) references users(username) PRIMARY KEY,
  average_rating numeric default 0 check(average_rating >= 0 AND average_rating <= 5)
);

CREATE TABLE owners (
  username varchar(64) references users(username) PRIMARY KEY
);

CREATE TABLE part_timers (
  username varchar(64) references caretakers(username) PRIMARY KEY,
  CHECK(is_full_timer(username)=0)
);

CREATE TABLE full_timers (
  username varchar(64) references caretakers(username) PRIMARY KEY,
  CHECK(is_part_timer(username)=0)
);

CREATE OR REPLACE FUNCTION is_full_timer(varchar) RETURNS NUMERIC AS $$
DECLARE ctx NUMERIC;
BEGIN
SELECT COUNT(*) INTO ctx FROM full_timers FT
WHERE $1 = FT.username;
IF ctx > 0 THEN
RETURN 1;
ELSE
RETURN 0;
END IF; END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION is_part_timer(varchar) RETURNS NUMERIC AS $$
DECLARE ctx NUMERIC;
BEGIN
SELECT COUNT(*) INTO ctx FROM part_timers PT
WHERE $1 = PT.username;
IF ctx > 0 THEN
RETURN 1;
ELSE
RETURN 0;
END IF; END;
$$ LANGUAGE plpgsql;

CREATE TABLE available_dates (
  username     varchar(64) references part_timers(username),
  start_period date NOT NULL,
  end_period   date NOT NULL,
  CHECK (start_period <= end_period),
  PRIMARY KEY(username, start_period, end_period)
);
```

```
CREATE TABLE leave_dates (
  username     varchar(64) references full_timers(username),
  start_period date NOT NULL,
  end_period   date NOT NULL,
  CHECK (start_period <= end_period),
  PRIMARY KEY(username, start_period, end_period)
);

CREATE TABLE pet_types (
  animal_name varchar(64) PRIMARY KEY,
  base_price  numeric NOT NULL
);

CREATE TABLE services (
  service_name varchar(64) PRIMARY KEY
);

CREATE TABLE handles (
  caretaker   varchar(64) references caretakers(username),
  animal_name varchar(64) references pet_types(animal_name),
  price       numeric     NOT NULL check(price >= getMinimumPrice(animal_name)),
  PRIMARY KEY(caretaker, animal_name)
);

CREATE FUNCTION getMinimumPrice(varchar)
RETURNS NUMERIC AS $$
 DECLARE price NUMERIC;
 BEGIN
      SELECT  base_price INTO price
      FROM    pet_types
      WHERE   animal_name = $1;

      RETURN price;
 END;
$$ LANGUAGE plpgsql;

CREATE TABLE provides (
  caretaker    varchar(64),
  animal_name  varchar(64),
  service_name varchar(64) references services(service_name),
  FOREIGN KEY(caretaker, animal_name) references handles(caretaker, animal_name),
  PRIMARY KEY(caretaker, animal_name, service_name)
);

CREATE TABLE pets (
  pet_name varchar(64),
  type varchar(64)  references pet_types(animal_name),
  owner varchar(64) references owners(username) ON DELETE CASCADE,
  PRIMARY KEY(pet_name, owner)
);

CREATE TABLE requires (
```

```
  owner varchar(64),
  pet_name varchar(64),
  service_name varchar(64)    references services(service_name),
  FOREIGN KEY(owner, pet_name) references pets(owner, pet_name),
  PRIMARY KEY(owner, pet_name, service_name)
);

CREATE TABLE bookings (
  owner varchar(64),
  pet_name varchar(64),
  caretaker varchar(64) references caretakers(username),
  start_period date,
  end_period date check(end_period >= start_period),
  payment_method varchar(64) NOT NULL,
  delivery_method varchar(64) NOT NULL,
  status varchar(64) NOT NULL,
  bid_rate numeric NOT NULL check(bid_rate >= getMinimumAskingPrice(caretaker, owner, pet_name)),
  rating numeric check(rating >= 0 AND rating <= 5),
  remarks varchar(1000),
  CHECK (start_period <= end_period),
  FOREIGN KEY(owner, pet_name) references pets(owner, pet_name),
  PRIMARY KEY(owner, pet_name, caretaker, start_period, end_period)
);
```

The constraints not captured by our schema are constraints 2, 4, 8, 14-22, 24. These have been covered by triggers. Specifically for constraints 2, 4:

- Constraint 2: *Covering constraint not met*
- Constraint 4: *Covering constraint not met; No-overlap constraint met with a checks calling functions displayed above part_timers and full_timers tables, namely is_full_timer(varchar) and is_part_timer(varchar) which check whether a new part-timer being added is already a full-timer or vice versa.*

Our Database is in BCNF

All primary keys have been underlined. The following tables have non-trivial functional dependencies. The tables not shown here don't have any non-trivial functional dependencies.

Table Name	Non - Trivial Functional Dependencies (Minimal Cover)
users	{ username -> password, username -> first_name, username -> location, username -> card_number}
pet_types	{ animal_name -> base_price }
handles	{ (caretaker, animal name) -> price }
pets	{ (pet_name, owner) -> pet_type }
bookings	{ (owner, pet_name, caretaker, start_period, end_period) -> payment_method, (owner, pet_name, caretaker, start_period, end_period) -> delivery_method, (owner, pet_name, caretaker, start_period, end_period) -> status, (owner, pet_name, caretaker, start_period, end_period) -> bid_rate, (owner, pet_name, caretaker, start_period, end_period) -> rating, (owner, pet_name, caretaker, start_period, end_period) -> remarks }

As can be seen from the table above, all the tables are in BCNF as the LHS only contains the primary key for all non-trivial functional dependencies. The union of all the functional dependencies of these tables gives us the functional dependencies of our database. Therefore, since the LHS only contains the primary key for all non-trivial functional dependencies, our database is in BCNF.

Interesting queries and triggers

Trigger 1: Constraint enforcement for leaves

This trigger ensures that various constraints for leaves are met when adding a new leave.
Firstly, it checks that there are no existing leaves that overlap with the new leave (first if block; for constraint 24).
Secondly, it checks that there are no existing accepted bookings within the leave period added (second if block; for constraint 14).

Lastly, it checks that the addition of the leave does not violate the 2x150 consecutive day per year constraint (constraint 8). This is done under the assumption that the current set of leaves do not already violate this constraint. The newly added leave only violates it if it ends up breaking up an existing period of 150/300 consecutive days.

In essence, we first determine the availability period that the new leave would break up by searching for the leaves flanking the newly added leave (or the start/end of the year), and determine whether the number of consecutive 150-day periods are conserved. The trigger checks for various cases. The grey and blue highlights represent cases where the leave crosses into another year and cases where the leave is within the same year respectively.

```
CREATE OR REPLACE FUNCTION check_insert_leave_full_timer() RETURNS trigger AS $ret$
  DECLARE year1 INT;
  DECLARE year2 INT;
  DECLARE date1 DATE;
  DECLARE date2 DATE;
  BEGIN
    IF (EXISTS(SELECT 1 FROM leave_dates
          WHERE username = NEW.username
           AND ((NEW.start_period <= start_period AND NEW.end_period >= start_period)
            OR (NEW.start_period BETWEEN start_period AND end_period)
            OR (NEW.end_period BETWEEN start_period AND end_period)))
      ) THEN RETURN NULL;
    END IF;

    IF (EXISTS(SELECT 1 FROM bookings
          WHERE EXISTS (SELECT 1 FROM full_timers WHERE NEW.username = full_timers.username)
           AND NEW.username = bookings.caretaker
           AND status = 'ACCEPTED'
           AND ((NEW.start_period <= start_period AND NEW.end_period >= start_period)
            OR (NEW.start_period BETWEEN start_period AND end_period)
            OR (NEW.end_period BETWEEN start_period AND end_period))))
      THEN RETURN NULL;
    END IF;

    year1 := EXTRACT(YEAR FROM NEW.start_period);
    year2 := EXTRACT(YEAR FROM NEW.end_period);

    IF (year1 < year2) THEN
      IF (year1 < year2 - 1) THEN
        RETURN NULL;
      END IF;
```

```
        IF (NEW.start_period - make_date(year1, 1, 1) < 300 OR make_date(year2, 12, 31) - NEW.end_period < 300)
        THEN  RETURN NULL;
        END IF;

        IF (EXISTS(SELECT 1 FROM leave_dates WHERE EXTRACT(YEAR FROM end_period) = year1) AND username =
            NEW.username) THEN
          SELECT end_period INTO date1 FROM leave_dates
          WHERE EXTRACT(YEAR FROM end_period) = year1
           AND username = NEW.username
          ORDER BY end_period DESC
          LIMIT 1;

          date2 := make_date(year1, 12, 31);
          IF (date2 - date1 >= 300 AND NEW.start_period - date1 < 301) THEN
            RETURN NULL;
          END IF;
          IF (date2 - date1 >= 150 AND NEW.start_period - date1 < 151) THEN
            RETURN NULL;
          END IF;
        END IF;

        IF (EXISTS(SELECT 1 FROM leave_dates WHERE EXTRACT(YEAR FROM end_period) = year2) AND username =
NEW.username) THEN
          SELECT start_period INTO date2 FROM leave_dates
          WHERE EXTRACT(YEAR FROM end_period) = year2
           AND username = NEW.username
          ORDER BY end_period ASC
          LIMIT 1;

          date1 := make_date(year2, 1, 1);
          IF (date2 - date1 >= 300 AND date2 - NEW.end_period < 301) THEN
            RETURN NULL;
          END IF;
          IF (date2 - date1 >= 150 AND date2 - NEW.end_period < 151) THEN
            RETURN NULL;
          END IF;
        END IF;

        RETURN NEW;
      END IF;


      date1 := make_date(year1, 1, 1);
      date2 := make_date(year1, 12, 31);

      IF (EXISTS(SELECT 1 FROM leave_dates
                        WHERE EXTRACT(YEAR FROM end_period) = year1
                          AND username = NEW.username
                          AND end_period < NEW.start_period))
            THEN date1 := (SELECT end_period + 1 FROM leave_dates
                WHERE EXTRACT(YEAR FROM end_period) = year1
                 AND username = NEW.username
```

```
                AND end_period < NEW.start_period
              ORDER BY end_period DESC
              LIMIT 1);
    END IF;

    IF (EXISTS(SELECT 1 FROM leave_dates
            WHERE EXTRACT(YEAR FROM end_period) = year1
              AND username = NEW.username
              AND start_period > NEW.end_period)) THEN
        date2 := (SELECT start_period - 1 FROM leave_dates
              WHERE EXTRACT(YEAR FROM start_period) = year1
                AND username = NEW.username
                AND start_period > NEW.end_period
              ORDER BY start_period ASC
              LIMIT 1);
    END IF;

    IF (date2 - date1 >= 149 AND NEW.start_period - date1 < 150
                                    AND date2 - NEW.end_period < 150) THEN
        RETURN NULL;
    END IF;

    IF (date2 - date1 >= 299 AND NEW.start_period - date1 < 150
                                    AND date2 - NEW.end_period < 300) THEN
        RETURN NULL;
    END IF;

    IF (date2 - date1 >= 299 AND NEW.start_period - date1 < 300
                                    AND date2 - NEW.end_period < 150) THEN
        RETURN NULL;
    END IF;

    RETURN NEW;
  END;
$ret$ LANGUAGE plpgsql;

CREATE TRIGGER insert_leave_full_timer
  BEFORE INSERT ON leave_dates
  FOR EACH ROW
  EXECUTE PROCEDURE check_insert_leave_full_timer();
```

Trigger 2: Direct Accepting of Bids for Full-Time Caretakers

This trigger was used to enforce the requirement that when any particular full-time caretaker has less than 60 pet days for any month in the duration of the booking, the booking would be automatically accepted for the caretaker. This is in direct relation to our 20th constraint.

To successfully implement this trigger, the function update_fulltimer_booking() was written. This function is responsible for checking the caretaker's existing bookings and accordingly setting "ACCEPTED" to the current booking's status in case it should be automatically accepted. The function handles this in three parts, over three 'AND' conditions:

1. The first operand of the AND condition (in Yellow) checks if the caretaker within the booking is indeed a full-timer, as automatic acceptance only applies to full-time caretakers.

2. The second operand of the AND condition (in Green) checks if in the months in which the current booking spans, whether the caretaker already has 60 petdays. This is done through the help of the generate_series built-in function which iterates over the start period of the booking to the end period of the booking on a month-by-month basis and checks if the petdays for any of them is greater than 60. If there is such a month within the start period and end period, then the booking should not be automatically accepted. Hence, the function checks for "NOT EXISTS" in this case.
3. The last operand of the AND condition (in Orange) uses the same generate_series built-in function to iterate over each day of the booking and check if in any of those days there exists a day on which the caretaker is already taking care of five pets. If there is such a day then there shouldn't be an automatic acceptance as the caretaker has already reached his limit. Hence, the function checks for "NOT EXISTS" in this case.

If all the above conditions are true, then it means the booking can be automatically accepted. This is what the "UPDATE" statement does at near the end of this function as it sets the status of the booking to "ACCEPTED".

```
CREATE OR REPLACE FUNCTION update_fulltimer_booking() RETURNS trigger as $ret$
  BEGIN
    IF NEW.caretaker IN (SELECT username FROM full_timers) AND
      NOT EXISTS (SELECT 1
                FROM (SELECT NEW.start_period + (interval '1' month * generate_series(0, CAST((DATE_PART('year',
new.end_period) - DATE_PART('year', new.start_period)) * 12 + (DATE_PART('month', NEW.end_period) -
DATE_PART('month', NEW.start_period)) AS INTEGER))) AS day) AS interval_months
                WHERE (SELECT SUM(
                        CASE
                        WHEN start_period > interval_months.day THEN
                        0
                        WHEN DATE_PART('month', interval_months.day) = DATE_PART('month', start_period) AND
(DATE_PART('month', interval_months.day) < DATE_PART('month', end_period) OR DATE_PART('year',
interval_months.day) < DATE_PART('year', end_period)) THEN
                        DATE_PART('day', (date_trunc('month', start_period) + interval '1 month') - start_period)
                        WHEN DATE_PART('month', interval_months.day) = DATE_PART('month', end_period) AND
(DATE_PART('month', interval_months.day) > DATE_PART('month', start_period) OR DATE_PART('year',
interval_months.day) > DATE_PART('year', end_period)) THEN
                        DATE_PART('day', end_period)
                        WHEN DATE_PART('month', start_period) = DATE_PART('month', interval_months.day) AND
DATE_PART('month', end_period) = DATE_PART('month', interval_months.day) AND DATE_PART('year',
start_period) = DATE_PART('year', end_period) THEN
                        end_period - start_period + 1
                        WHEN DATE_PART('month', start_period) < DATE_PART('month', interval_months.day) AND
DATE_PART('month', end_period) > DATE_PART('month', interval_months.day) AND DATE_PART('year',
start_period) <= DATE_PART('year', interval_months.day) AND DATE_PART('year', end_period) >=
DATE_PART('year', interval_months.day) THEN
                        DATE_PART('day', date_trunc('month', interval_months.day) + interval '1 month' - interval '1
day')
                        WHEN DATE_PART('month', interval_months.day) > DATE_PART('month', end_period) AND
DATE_PART('year', end_period) > DATE_PART('year', interval_months.day) THEN
                        DATE_PART('day', date_trunc('month', interval_months.day) + interval '1 month' - interval '1
day')
                        WHEN DATE_PART('month', interval_months.day) < DATE_PART('month', end_period) AND
DATE_PART('year', end_period) > DATE_PART('year', interval_months.day) THEN
                        DATE_PART('day', date_trunc('month', interval_months.day) + interval '1 month' - interval '1
day')
                    END)
```

```
                FROM (SELECT * FROM bookings EXCEPT SELECT * FROM bookings WHERE NEW.owner=owner
AND NEW.pet_name=pet_name AND NEW.caretaker=caretaker AND NEW.start_period=start_period AND
NEW.end_period=end_period) AS b
                WHERE caretaker=NEW.caretaker AND status='ACCEPTED') >= 60) AND
        NOT EXISTS (SELECT *
                FROM (SELECT NEW.start_period + (interval '1' day * generate_series(0, (CAST((NEW.end_period -
NEW.start_period) AS INTEGER)))) AS days) AS dates
                WHERE (SELECT COUNT(*) FROM bookings b WHERE b.caretaker=NEW.caretaker AND
b.start_period<=dates.days AND b.end_period>=dates.days AND "status"='ACCEPTED') = 5)
    THEN UPDATE bookings SET "status" = 'ACCEPTED' WHERE NEW.owner=owner AND
NEW.pet_name=pet_name AND NEW.caretaker=caretaker AND NEW.start_period=start_period AND
NEW.end_period=end_period;
    END IF;
    RETURN NEW;
  END;
$ret$ LANGUAGE plpgsql;

CREATE TRIGGER direct_accept
AFTER INSERT ON bookings
FOR EACH ROW
EXECUTE PROCEDURE update_fulltimer_booking();
```

Trigger 3: Automatic Declining of Clashing Bids

This trigger was used to enforce the idea that after a booking is accepted by the caretaker, any booking for that caretaker which then clashed (could no longer be accepted as the caretaker has already reached his limit for pets for those days) have their status accordingly set to “DECLINED”. This was done through the “decline_clashing_bids” trigger which calls the function “decline_clashing()” when the “status” attribute of a booking is changed. The “decline_clashing()” function works as follows for full-time caretakers:

1. The first if condition (in Blue) checks if the new status is “ACCEPTED”. Only then will the actions of this trigger take place.
2. The second if condition (in Yellow) checks whether the caretaker is a full-timer. Conversely, the else condition handles the situation of if the caretaker is otherwise a part-timer. The function then updates the status of all bookings corresponding to the particular caretaker from “PENDING” to “DECLINED” where certain conditions (explained below) are met.
3. The function then iterates one day at a time from the start period of the booking to the end period of the booking to be updated (in Orange). If for any of those days, there already exist 5 “ACCEPTED” bookings (the limit for full-timers) in the name of the particular caretaker it means the condition is met and this booking should be automatically declined, as the “UPDATE” keyword does.

A similar procedure is carried out for part-time caretakers as well. However, to incorporate the idea that part-timers with a rating of more than 4.0 can take care of up to 4 pets, there is an additional query to return the maximum number of pets they can take care of at a time.

```
CREATE OR REPLACE FUNCTION decline_clashing() RETURNS trigger AS $ret$
  BEGIN
    IF new.status = 'ACCEPTED' THEN
      IF EXISTS (SELECT 1 FROM full_timers WHERE username=NEW.caretaker)
        THEN UPDATE bookings b1 SET "status" = 'DECLINED' WHERE "status"='PENDING' AND
            caretaker=NEW.caretaker AND
            EXISTS (SELECT 1
                FROM (SELECT b1.start_period + (interval '1' day * generate_series(0, CAST((b1.end_period -
b1.start_period) AS INTEGER))) AS days) AS dates
```

```
                    WHERE (SELECT COUNT(*) FROM bookings b WHERE b.status='ACCEPTED' AND
b.caretaker=NEW.caretaker AND b.start_period<=dates.days AND b.end_period>=dates.days) = 5);
      ELSE
        UPDATE bookings b2 SET "status" = 'DECLINED' WHERE "status"='PENDING' AND
              caretaker=NEW.caretaker AND
              EXISTS (SELECT 1
                  FROM (SELECT b2.start_period + (interval '1' day * generate_series(0, CAST((b2.end_period -
b2.start_period) AS INTEGER))) AS days) AS dates
                  WHERE (SELECT COUNT(*) FROM bookings b WHERE b.caretaker=NEW.caretaker AND
b.start_period<=dates.days AND b.end_period>=dates.days) = (SELECT CASE WHEN (SELECT average_rating FROM
caretakers WHERE username=NEW.caretaker) >= 4.0 THEN 4 ELSE 2 END));
        END IF;
    END IF;
  RETURN NEW;
 END;
$ret$ LANGUAGE plpgsql;
```

Interesting Queries

Query#1: Profit calculation

This query calculates the total profit for a particular month. For a booking to be in a particular month, there are 4 cases - it starts before the month and ends before the month, starts before the month and ends in the month, starts in the month and ends after the month, starts in the month and ends in the month. We get the number of pet days in the month accordingly. These are the 4 cases considered in date calculations used in different parts of the query. The first part of the query (highlighted in yellow), calculates total revenue in the month as the sum of bid rate * number of pet days for each accepted booking of the month. From the revenue, we subtract the sum of the salary which we get from a subquery in the FROM section. For salary itself, we need to consider 3 cases - part timers(highlighted in orange), full timers who have worked over 60 pet days that month (highlighted light green) and full timers that have worked less than 60 pet days that month (highlighted in blue). Part timers earn sum of (0.75 * number of pet days * bid rate of booking)$ for all of their bookings. Full timers who have worked over 60 pet days earn (3000 + 0.8 * (total number of pet days - 60) * average bid rate)$. Average bid rate is calculated as sum(bid rate * pet days) for each booking / total number of pet days so that the bid rate of longer bookings can be given more weightage. Full timers who have worked 60 pets days or under earn 3000$. This query could have been simplified with the use of CTE but we didn't use one due to the project constraints. $1 is the date in the month whose profit we want to find out.

```
SELECT (SELECT SUM( CASE
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date) *
bid_rate
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', end_period::date) * bid_rate
    WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
        (end_period::date - start_period::date + 1) * bid_rate
    WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
```

```
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
        DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day') *
bid_rate
  END)
FROM bookings
WHERE status = 'ACCEPTED') - SUM(salary) AS profit
FROM (SELECT C.username AS cusername, COALESCE((SELECT SUM(
  CASE
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date) *
bid_rate * 0.75
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', end_period::date) * bid_rate * 0.75
    WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
        (end_period::date - start_period::date + 1) * bid_rate * 0.75
    WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
        DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day') *
bid_rate * 0.75
  END)
  FROM bookings
  WHERE caretaker = C.username AND status = 'ACCEPTED'), 0) AS salary
FROM part_timers C
WHERE C.username IN (SELECT username FROM part_timers)
UNION
SELECT C.username AS cusername, 3000 + 0.8 * (SELECT SUM(
  CASE
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date) *
bid_rate
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', end_period::date) * bid_rate
    WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
        (end_period::date - start_period::date + 1) * bid_rate
    WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
```

```
        DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day') *
bid_rate
  END)
FROM bookings
WHERE caretaker = C.username AND status = 'ACCEPTED')/(SELECT SUM(
  CASE
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date)
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', end_period::date)
    WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
        (end_period::date - start_period::date + 1)
    WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
        DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day')
  END)
FROM bookings
WHERE caretaker = C.username AND status = 'ACCEPTED') * ((SELECT SUM(
  CASE
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date)
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', end_period::date)
    WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
        (end_period::date - start_period::date + 1)
    WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
        DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day')
  END)
FROM bookings
WHERE caretaker = C.username AND status = 'ACCEPTED') - 60) AS salary
FROM full_timers C
WHERE C.username IN (SELECT username FROM full_timers) AND (
SELECT SUM(
  CASE
```

```
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date)
    WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', end_period::date)
    WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
        (end_period::date - start_period::date + 1)
    WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
        DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day')
  END)
FROM bookings
WHERE caretaker = C.username AND status = 'ACCEPTED') > 60
UNION
SELECT C.username AS cusername, 3000 AS salary
FROM full_timers C
WHERE C.username IN (SELECT username FROM full_timers) AND (SELECT SUM(
CASE
WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
  DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date)
WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
  DATE_PART('day', end_period::date)
WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
  (end_period::date - start_period::date + 1)
WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
  DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day')
END)
FROM bookings
WHERE caretaker = C.username AND status = 'ACCEPTED') <= 60
  OR (SELECT SUM(
    CASE
      WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', start_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) < DATE_PART('month', end_period::date) OR DATE_PART('year',
CAST( $1 AS TIMESTAMP )) < DATE_PART('year', end_period::date)) THEN
        DATE_PART('day', (date_trunc('month', start_period::date) + interval '1 month') - start_period::date)
```

```
      WHEN DATE_PART('month', CAST( $1 AS TIMESTAMP )) = DATE_PART('month', end_period::date) AND
(DATE_PART('month', CAST( $1 AS TIMESTAMP )) > DATE_PART('month', start_period::date) OR
DATE_PART('year', CAST( $1 AS TIMESTAMP )) > DATE_PART('year', end_period::date)) THEN
          DATE_PART('day', end_period::date)
      WHEN DATE_PART('month', start_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) = DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) = DATE_PART('year', end_period::date) THEN
          (end_period::date - start_period::date + 1)
      WHEN DATE_PART('month', start_period::date) < DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('month', end_period::date) > DATE_PART('month', CAST( $1 AS TIMESTAMP )) AND
DATE_PART('year', start_period::date) <= DATE_PART('year', CAST( $1 AS TIMESTAMP )) AND DATE_PART('year',
end_period::date) >= DATE_PART('year', CAST( $1 AS TIMESTAMP )) THEN
          DATE_PART('day', date_trunc('month', CAST( $1 AS TIMESTAMP )) + interval '1 month' - interval '1 day')
    END)
  FROM bookings
  WHERE caretaker = C.username AND status = 'ACCEPTED') IS NULL) AS salaries
```

Query#2: Get Month with Max Jobs/Max Pets Taken Care Of

This query returns the month in which there are the maximum number of jobs of both full-time and part-time caretakers. The process of counting the number of pets taken care of each month is subdivided into three parts that return different count values. The first is to calculate the caretakers whose job started in a particular month. The second is for the jobs that ended in the particular month. A check is made to ensure that the start of the month is not the same as the end so that double counting is not made. Then for the jobs that start on a month and end after >1 month, a generate series function is used to generate temporary dates for the months in between. Hence all these different combinations are listed in one table and a sum of the total count is made for each month. Then finally, the months were rearranged in descending order so that the month with maximum jobs is the first tuple and then we limit 1 to get the first entry.

```
SELECT month
  FROM
    (SELECT SUM(count1) AS jobs, month
    FROM
      (SELECT COUNT(*) AS count1, DATE_PART('month', b1.start_period) AS month
      FROM bookings b1 GROUP BY DATE_PART('month', b1.start_period)
    UNION ALL
      SELECT COUNT(*) AS count2, DATE_PART('month', b1.end_period) AS month
      FROM bookings b1
      WHERE DATE_PART('month', b1.start_period) <> DATE_PART('month', b1.end_period)
      GROUP BY DATE_PART('month', b1.end_period)
    UNION ALL
      SELECT COUNT(*) AS count3, DATE_PART('month', month) AS month
      FROM (SELECT generate_series(date_trunc('month',  start_period), end_period, '1 month')::date as month
          FROM bookings
          WHERE  DATE_PART('month', end_period) - DATE_PART('month', start_period) > 1) as temp
          GROUP BY DATE_PART('month', month)) as temp
    GROUP BY month) as ans
  ORDER BY jobs DESC
  LIMIT 1;
```

Query #3: Caretaker search

This query returns the username, first name, and price of available caretakers that can provide all required services to a specified pet (by owner's username and pet's name) over a specified period of time. First, we only select part-time caretakers who have specified that they are available during the period of time, and caretakers who are

not on leave during that period of time (grey portion). This assumes that for part-time caretakers, consecutive periods of availability are merged, which we handle in a separate trigger not in this report.

In the blue highlighted portion, we check that there is no day during the specified period where the number of pets already taken care is at or above the limit [5 for full-timers, 2(or 4) for part-timers (with rating above 4)]. Lastly, we check that the caretaker does indeed handle the pet type, and that they provide all services required by the specific pet (green portion).

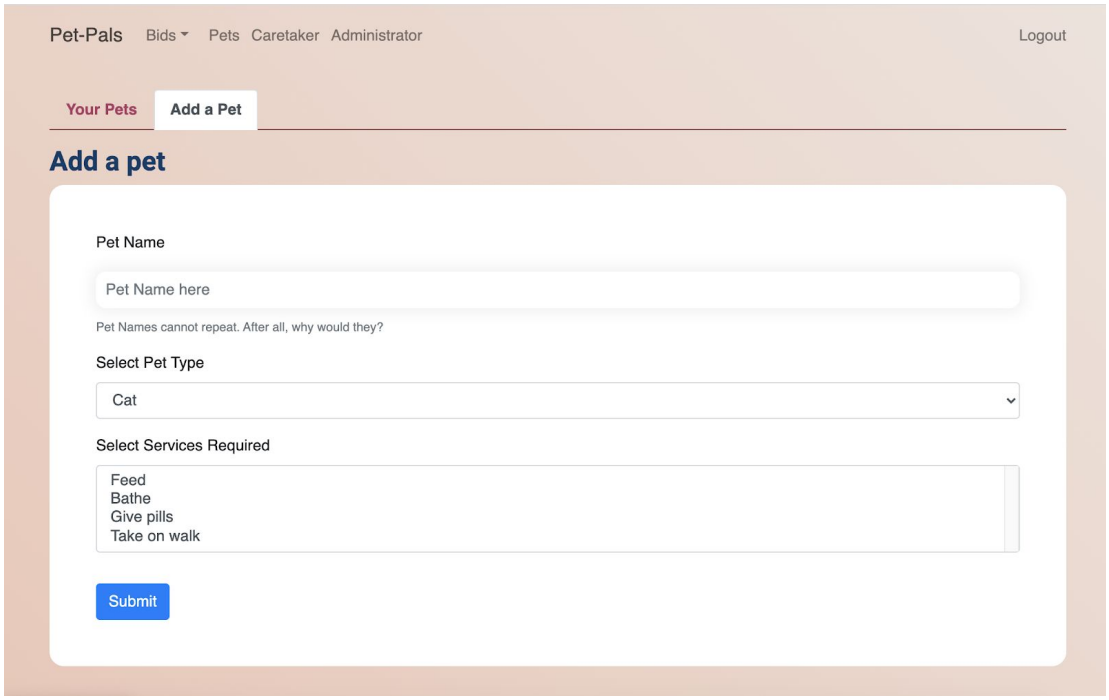
```
SELECT U.username, U.first_name, H.price FROM users U, handles H WHERE CAST($1 AS DATE) <= CAST($2 AS
DATE)
  AND ((EXISTS(SELECT 1 FROM full_timers F WHERE F.username = U.username)
            AND NOT EXISTS(SELECT 1 FROM leave_dates L WHERE L.username = U.username
                  AND (L.start_period BETWEEN CAST($1 AS DATE) AND CAST($2 AS DATE)
                    OR L.end_period BETWEEN CAST($1 AS DATE) AND CAST($2 AS DATE)
                    OR (L.start_period <= CAST($1 AS DATE) AND L.end_period >= CAST($1 AS DATE))
                    )
            )
        )
       OR (EXISTS(SELECT 1 FROM part_timers P WHERE P.username = U.username)
          AND EXISTS(SELECT 1 FROM available_dates A
                WHERE A.start_period <= CAST($1 AS DATE)
                  AND A.end_period >= CAST($2 AS DATE)
                  AND A.username = U.username)
        )
  )
  AND NOT EXISTS (SELECT CURRENT_DATE + i
            FROM generate_series(CAST($1 AS DATE) - CURRENT_DATE, CAST($2 AS DATE) - CURRENT_DATE) i
              WHERE (SELECT COUNT(*) FROM bookings B
                  WHERE B.status = 'ACCEPTED'
                  AND B.caretaker = U.username
                  AND CURRENT_DATE + i BETWEEN B.start_period AND B.end_period) >=
                  (SELECT CASE
                    WHEN EXISTS(SELECT 1 FROM full_timers F WHERE F.username = U.username) THEN 5
                    WHEN (SELECT C.average_rating FROM caretakers C WHERE C.username = U.username) >= 4
                          THEN 4
                    ELSE 2
                    END
                  )
  )
  AND H.animal_name = (SELECT type FROM pets WHERE pet_name = $4 AND owner = $3)
  AND H.caretaker = U.username
  AND NOT EXISTS (SELECT 1 FROM requires R
            WHERE R.pet_name = $4 AND R.owner = $3
            AND NOT EXISTS (SELECT 1 FROM provides P
                        WHERE P.caretaker = U.username
                        AND P.service_name = R.service_name
                        AND P.animal_name = (SELECT type FROM pets WHERE pet_name = $4 AND owner = $3)
                        )
            )
```

Software Tools/Frameworks Used:

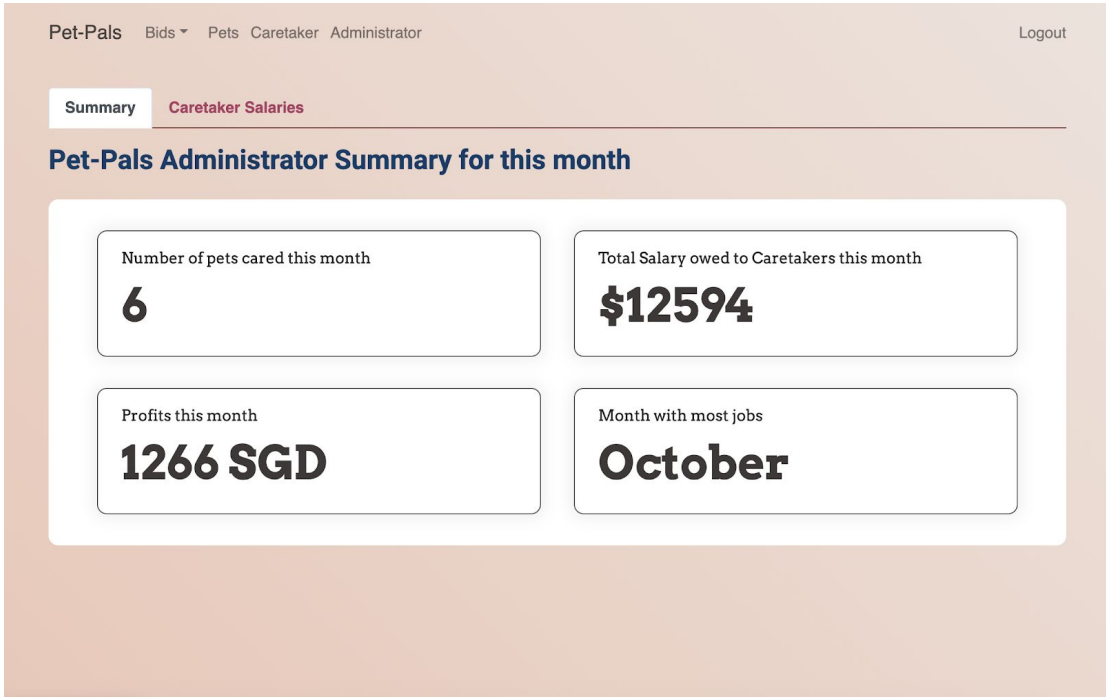
Frontend - React.JS
Backend - Node.JS
Database Engine - PostgreSQL

User Interface

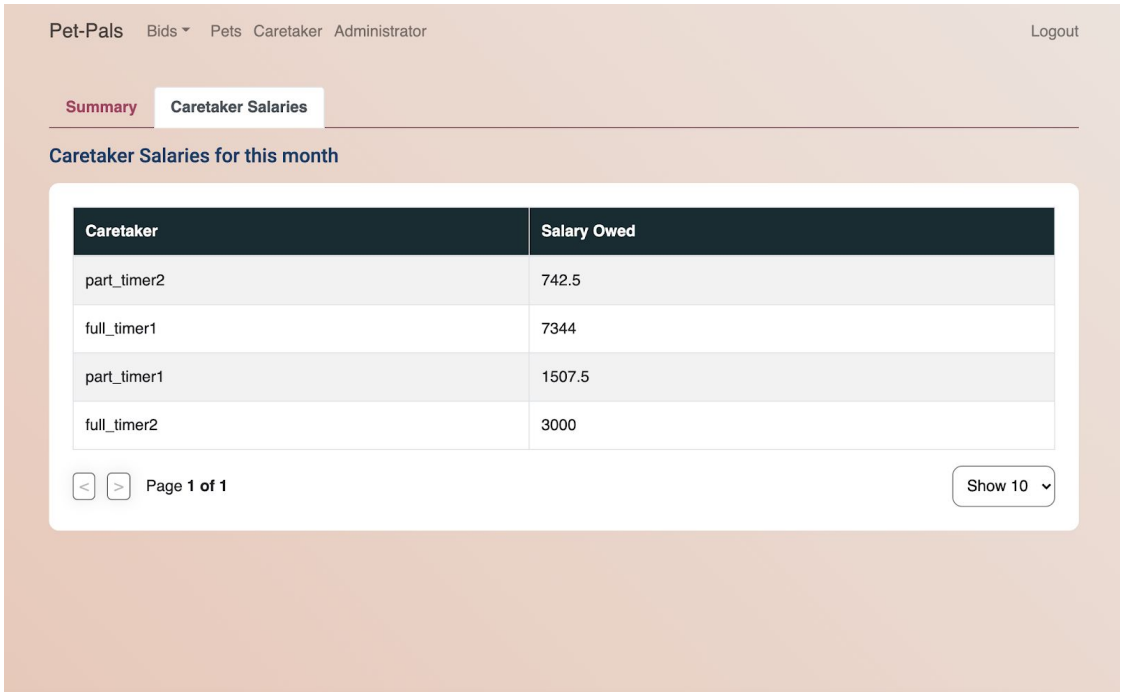
Adding a New Pet as a Pet Owner



Viewing the Statistics of the Current Month as an Administrator



Viewing all Salaries to be Paid in the Current Month as an Administrator



Difficulties Encountered & Lessons Learned

- This was the first time working on a web application for most of us, so we had to spend some time figuring out how things work.
- The requirements given were often vague and we had made several design decisions in the beginning that made a huge impact on our work later on. This experience taught us to be more far-sighted when making decisions in the beginning.
- Queries regarding dates were quite challenging as we had to consider a lot of cases regarding start dates and end dates (as can be seen in our 'interesting' queries).
- Some of our constraints required checks across tables which we didn't initially know how to do since it wasn't taught in the course.
- Working on the UI was also quite challenging due to our inexperience with UI work but this helped us develop it.
- We worked on the database queries, backend and frontend separately and had a challenging time integrating them together. We learned to appreciate the agile workflow more.