CS2102 AY2020/2021 Sem 1 Project Report

Prepared by Group 45

Project Members	Responsibilities
1. Khiew Zhi Kai (A0167442A)	A. Coding SQL queries, triggers, schema. B. Drawing up the ER diagram. C. Generate test cases used in the testing of the queries.
2. Yan Boshen (A0190040X)	A. Usage of NodeJS in sending and receiving queries from front end and back end. B. Generate data. C. Writing of SQL queries.
3. Wong Jun Hong (A0182774U)	A. Front end design and functionalities.
4. Vanessa Tan Li Xuan (A0185493R)	A. Coding SQL queries and schema. B. Generate test cases used in the testing of the queries.
5. Li Huihui (A0188608N)	A. Coding SQL queries and schema. B. Generate test cases used in the testing of the queries.

1. Data Requirement and Functionalities

1.1 Functionalities

Interesting, non-trivial aspects of our application

Browsing of available caretakers between the date range specified

Ability for PetOwners to obtain the list of caretakers who are available for every single day in the date range specified by him/her. The query from the PetOwner will consider the desired start date of service, end date of service and pet type. It will return a list of caretakers who are available to take up the job for every single day between the start date and end date, the per day fee of the caretaker's service, the review and ratings of the caretaker. Query will consider:

1. The maximum number of pets that the caretaker can take care of per day, and whether the quota is reached. If quota has been reached for any day within the date range, this caretaker will not be listed as a choice in the query result.
2. Whether caretaker can take care of the pet type in query.

Only caretakers that are available for entire service period are chosen, so that the pet does not have to be transferred to another caretaker in between the service period. Viewing of fee per day, ratings and review helps PetOwner make informed decision.

Computation of salary for the month

The FullTime caretaker will have a monthly base pay of $3000. If the FullTime caretaker has worked for more than 60 pet days in the month, the income from the excess pet days will be multiplied by 0.8 and added to his base pay. The lowest 60 pet day price will be used for the base 60 pet days, and the remaining will be considered excess pet days. The salary of the PartTime caretakers is computed using the income from all the pet days he/she had taken up multiplied by 0.75.

The caretakers and the PCS admin will be able to view the salary. The method of calculation for FullTime caretaker's salary benefits the FullTime caretaker by maximising his bonus for excess pet days.

Ability to view bidding status for bids that were rejected as caretaker went on leave after the bid was placed by PetOwner
Introduced a table called InvalidatedBids (discussed in non-trivial ER design section 2.1.3)
This feature allows the PetOwner involved in such transactions to be able to view that his bid was rejected, instead of the bid getting deleted without notice to the PetOwner who placed the bid.

The ability for PetOwners, caretakers, PCS Admins to delete their account.
Triggers were implemented to facilitate smooth deletion of accounts, without losing essential information in the database. (discussed in Non-trivial ER design section 2.1.3, 2.1.4 and Interesting Triggers section 5.3)

Ability to view past bids by PetOwner who deleted account
Introduced a table called BidsWithoutPetOwner (discussed in non-trivial ER design section 2.1.4)
This prevents the loss of information from the Bids table when the PetOwner's account is deleted. The information of the past bids relating to deleted account is required for computing salary and average rating of the caretakers involved in the transactions.

Each new caretaker must have been hired by the company before he/she can sign up as a new caretaker.
This requirement is enforced by the requirement to provide the PCS Admin's username during creation of a new caretaker account.
Upon successful job offer, the new caretaker will obtain the username of the PCS Admin that manages him/her.
Helps prevent unsuitable candidates from becoming a caretaker, also ensures every caretaker is managed by a PCS Admin.

Ability for each caretaker to adjust their base fee per day for caretaking service according to their average rating.
Caretakers can set their base fee per day for each pet type. The lower limit of the base fee per day of each pet type that the Caretakers can set would be the base price specified by the company. The upper limit of the settable base fee per day depends on the caretaker's average rating. (mentioned in Data Constraints 1.2.11) Caretakers thus have the incentive to obtain higher ratings.

Ability for the PetOwner to place bids on selected caretaker for pet caring service.
The PetOwner can place a bid on his/her selected caretaker from the query results. The bid placed must be equals to or above the base fee per day set by the chosen caretaker. The bid placed reflects the amount of money the PetOwner is willing to offer for each day of the caretaker's service. The caretaker can select his customers according to the rates offered by the PetOwners.

Automatic rejection of all other bids for the same service when a bid is accepted by a caretaker
All the other bids placed for the service in the same date range will be automatically rejected. This is implemented as a trigger. (mentioned in Data Constraints 1.2.20d) This prevents multiple acceptance of the same service by different caretakers.

Other basic functionalities of our application
Create an account (PetOwner, CareTaker, both or PCS admin); PetOwner leaving review and rating for caretaker after completion of a service; Supports viewing bids offered, current base price of pet categories by CareTakers; view ratings of all CareTakers.

1.2 Data Requirements:

The following are all the *data constraints* required in our application.

1) Users are uniquely identified by their User Name. Their Profile (words to describe oneself), Password, Area/Address, Phone Number must be provided. Email is optional. User Name must be unique to others.
2) There are two types of Users,
    a. They are either Consumers or PCS Admin(Covering Constraint)
    b. And cannot be both (Non-Overlapping Constraint).
3) There are two types of Consumers,
    a. They are either PetOwner or CareTaker (Covering Constraint).
    b. PetOwners can be CareTakers as well (Overlapping Constraint).
    c. Each Caretaker is managed by exactly one a PCS Admin account.
4) Consumers may provide their Bank Account Number to pay for any transaction or to credit their salary. They may also provide one credit card for payment of services.
5) Pets are uniquely identified by their PetName and PetOwner, and must have a profile. Optionally, Special Requirements of the pet may be added as an attribute. Each Pet is owned by exactly one PetOwner. A PetOwner account may have zero or more pets registered, although to bid for care-taking services requires a pet.
6) The existence of a pet on the PCS application depends on the PetOwner, once the PetOwner is deleted, we need not keep track of the Pet under that owner any longer. A Pet cannot uniquely identify their owners.
7) Each Pet should belong to exactly one valid Pet Type.
8) Pet Types are identified by their Category (e.g dog, cats), and each category must have a minimum base price assigned by the PCS Admin.
9) There are two types of CareTakers: Full Timer and PartTimer. They must be either one of each type and cannot be both types (Covering Constraint).
    a. All Full Timers have a max pet limit of 5.
    b. PartTimers have a max pet limit between 2 to 5 which is dependent on the average rating of the PartTimer. If the average rating of the PartTimer is above 3 but below or equals to 4, limit is set to 3. If the average rating is above 4

and below or equals to 4.7, the limit is set to 4. If the rating is above 4.7, the limit is set to 5. If the rating of the PartTimer drops, resulting in a decrease in limit, successful bookings that exceeds the new limit will be kept. However, all future bids and approval of bids will be subjected to the limit.

10) CareTakers have to be able to take care of at least one Pet Type.
11) Each CareTaker will also have a Fee per Day specified (by them) for the services of caring for each Pet Type.
    a. This fee is greater than or equals the base price of the category of Pet Type.
    b. Fee per Day can be priced up to 1.1 times of base price if the CareTaker has an averaged rating of 4 or higher, and up to 1.2 times of base price if the CareTaker has an average rating of 4.7 or higher as the upper limit, otherwise the upper limit is equal to the base price.
12) When the PCS Admin updates the base price for a particular pet category, check if the existing fee per day for any caretaker for that category exceeds the upper limit, or falls below the base price. If so, reset to new base price.
13) CareTakers' available Dates (working days) will be tracked over the course of the next two years. Once Caretaker is deleted, we need not track their availability.
14) FullTime CareTakers are assumed to be available every day for the whole of the next two years, unless they decide to take leave on a particular Date.
15) PartTime CareTakers have to update their available dates for the next two years.
    a. They also cannot apply for leave.
16) FullTimers can apply for Leave Days to be taken. These Leave days allow their Availability dates (working days) to be removed from consideration, their Available dates for the year will be tracked (their leave dates are simply dates not in Available dates). There are two conditions that have to be satisfied for the new Leave to be approved.
    a. They have no Pet booked to care for during that date.
    b. Their leave should not violate the minimum 2x150 consecutive days of work for the year.
17) A Bidding Record should be identifiable by the PetOwner, PetName, CareTaker User Name, and the Date of Service and the CareTaker involved. It must include information on the Transfer Type, Payment Type, End Date of service, and a Bid Price. There must also be an attribute isPaid, indicating if the PetOwner has paid for the service.
18) Each Bidding Record has a compulsory attribute called Status. Status can take three possible values: Approved, Rejected or Pending.
    a. When the PetOwner first bids for a CareTaker's service on behalf of the pet, he or she will have to fill in the required fields, and then it will have the 'pending' status.
    b. Once the caretaker accepts the bid, the status of the bidding record would be updated to 'approved' status.
    c. Likewise, if the caretaker rejects the bid, the bidding record would be updated to 'rejected' status.
19) PetOwners may bid for CareTaker's services on behalf of their Pet on the days the particular CareTaker is available.
    a. PetOwners can only bid for available dates in the future (after current date)
    b. A PetOwner may bid for a range of consecutive dates for the Available dates of the CareTaker.
        i. This will create individual bid entries for each of the consecutive Available dates bidded for, with the same values for all the other attributes for the bidding record.
        ii. These multiple bids for consecutive dates should all have the same End Date as an indication that the bid for these multiple bids across a range of dates should be considered together as a single transaction.
    c. A CareTaker's services are only open for bidding for a new bid, if they are Available(working) on a particular day and have at least 1 remaining pet slot.
    d. The PetOwner should also bid only for CareTakers that can care for their Pet's Type for their bid to be valid.
    e. If the Pet of a particular PetOwner already has an approved bid for a given date, he/she cannot bid for anymore services for that same pet on that same date.
    f. The new bid has to include information on the Transfer Type (either handing the pet over via Owner Delivery, CareTaker pick up or PCS Center), and the Payment Type (credit card or cash) when they bid.
    g. Every bid must have a bid price greater than or equal to the CareTaker's fee per day for that pet category.
    h. Each bid will appear in the Bidding Records with a pending status initially.
    i. Each bid will initially have null values for review and rating.
    j. Users who are both caretaker and PetOwners may not bid for their own services.
20) CareTakers will review the bids and accept the bids according to his preference.
    a. Updating a bid to approved status by the CareTaker should only be for bids on dates after the current date.
    b. A bid can only be approved by the CareTaker if he/she is available to take at least one more pet without exceeding his/her max pet limit.
    c. A CareTaker can only reject or accept all the bids from a Pet and PetOwner within a single transaction as a group, that is, the set of bids with a common end date.
    d. Once a bid is accepted by a CareTaker, all bids by the same PetOwner, with the same Pet on the same date for other caretakers will be rejected automatically. This rejection is then applied to the entire transaction.

    e. If a caretaker becomes fully booked after approving a bid, all other bids pending for this caretaker on that date should be rejected as well.
    f. If a bid has been rejected, it cannot be approved later.

21) The Bidding Records could have a Rating (integer from 1 to 5) and Review (string of text) by the PetOwner regarding the service provided by the CareTaker.
    a. The Rating and Review can be filled by the PetOwner only for approved bids.
    b. Once a rating is set, it cannot be changed anymore.
    c. There can only be one review and rating for each complete transaction (defined as a range of consecutive days of service for a particular pet) with a CareTaker.
    d. If the update of a rating causes the average rating of a CareTaker to fall such that their upper bound on their fee per day for each pet type falls, check if their existing fee per day exceeds the upper bound and reset it to base price if so.
    e. If the update of a rating causes the max pet limit of a PartTime CareTaker to fall (due to lower average rating), then some previously available dates (for booking a service) are no longer available, reject all pending bids for the unavailable dates.

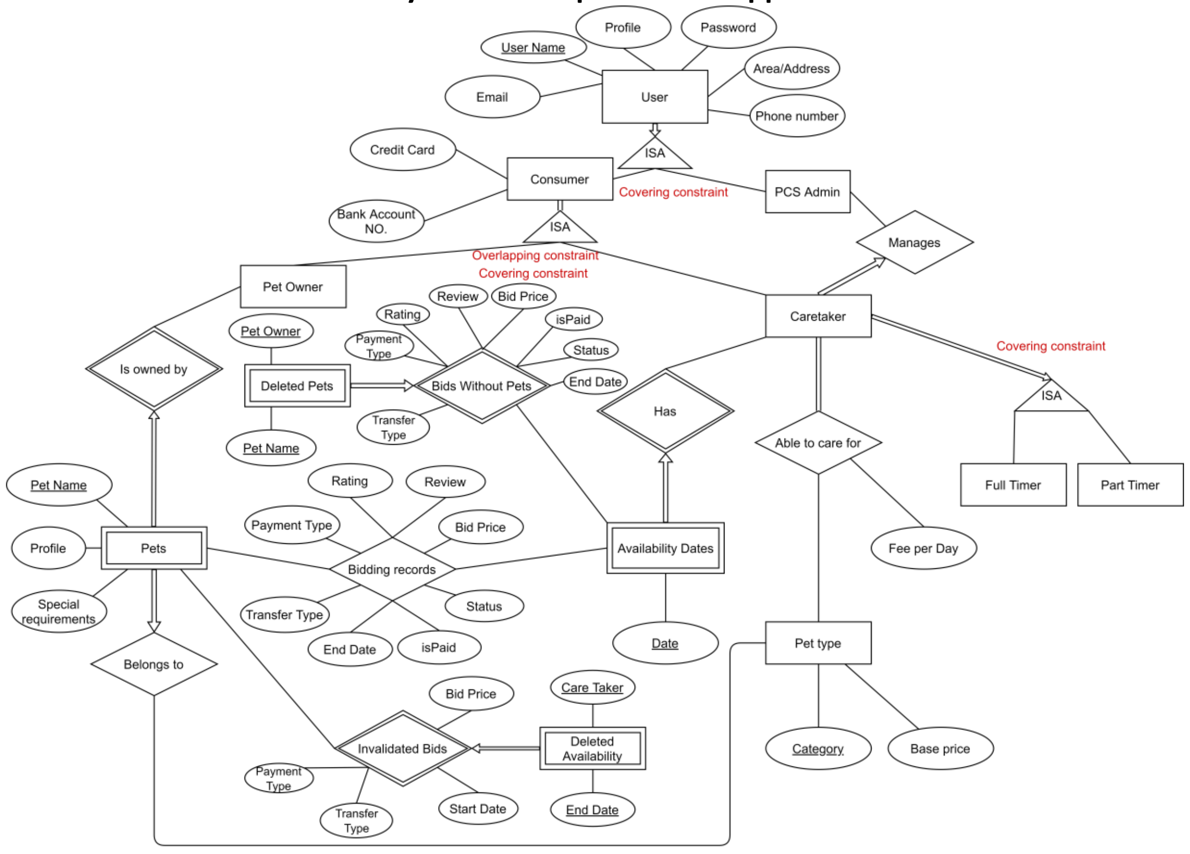
22) When leave is successfully applied, or CareTaker account is deleted, pending or rejected bids on the affected Availability dates should have its data saved in an Invalidated Bids table as a single transaction before being deleted from Bidding Records.
    a. The Invalidated bids table has PetOwner username, PetName, CareTaker and End Date (of transaction) as primary keys, Start Date, Transfer Type, Payment Type and Bid Price as other fields.
    b. Once the Pet or PetOwner is deleted, we need not track the transaction in this table anymore.

23) When a Pet or PetOwner account is deleted, the affected bids should be transferred to a BidsWithoutPetOwner table. Because rating, review and bid price are still important to be tracked for the CareTaker it should be regarded as part of the Bidding Records.
    a. The BidsWithoutPetOwner table has the same primary key and attributes as Bids table.
    b. However, once the Caretaker is deleted, we need not track the bid in this table anymore.

24) A particular Pet, PetOwner account or CareTaker account cannot be deleted if there are still approved bids in the future (after the current date) yet to be completed. (This guarantees each entry to BidsWithoutPetOwner has a unique primary key because the date will be different even if a PetOwner with the same username is deleted and created multiple times)

25) The PCS Admin account cannot be deleted if there are caretakers being managed by that admin account.

2. Entity Relationship Model of Application



ER Diagram: Refer to Annex 3.1 For Full Page Diagram

2.1 Non Trivial Designs in the Entity-Relationship Model

1) Description of Availability Table

In Availability table, for example, if caretaker Lisa is working on 2020-01-01 to 2020-01-02, there will be two entries in the table: ('Lisa', '2020-01-01') and ('Lisa', '2020-01-02'). When FullTime caretakers apply for leave, the entries with the leave dates will be removed from Availability.

2. Description of Bids Table

Our Bids Table is designed such that for every date in one bid, there will be a row in the Bids Table. The entries are of form: (PetOwner username, PetName, caretaker username, date, end date, transfer type, payment type, price, isPaid, status, review, rating)

Date refers to date in each day of transaction period. An example of bidding for Caretaker 'Lisa' by PetOwner 'Claire' for her pet 'Munchy' for a period between 2020-01-01 to 2020-01-02 would be:

('Claire', 'Munchy', 'Lisa', '2020-01-01', '2020-01-02', 'PCS building', 'credit card', 15, False, 'p', NULL, NULL)
('Claire', 'Munchy', 'Lisa', '2020-01-02', '2020-01-02', 'PCS building', 'credit card', 15, False, 'p', NULL, NULL)

Every transaction is identified by the PetOwner's username, PetName, Caretaker's username, and end date of the transaction. Each bidding record has a compulsory attribute called Status which takes value: 'a' for accepted, 'r' for rejected and 'p' for pending. When bid is first placed, status of every entry of bid would be 'p'. Acceptance of bid will update the status of each entry of the transaction to 'a'. Vice versa for rejection of bid. Finished transactions will be identified by status = 'a' and isPaid is True. Presenting each day of caretaking service allows easier computation, where the number pet days served is easily accessible by COUNT() and total service fee can be accessed by SUM(). The PetOwners would be able to leave a review for their transaction after transaction is successfully completed. For every transaction, only the last date of transaction in the Bids table will be updated with the rating and review. This makes it easier to calculate average rating for caretakers and avoiding repetition of duplicate data.

3. Description of InvalidatedBids Table

A caretaker may decide to take a leave on a day he was listed as available or delete his account. There may be pending bids for that caretaker on the days he takes leave or delete his account. Due to the delete cascade from Availability table, the transaction record will be deleted from the Bids table. To allow PetOwners to view such invalidated bids, we incorporated the InvalidatedBids table where its entries take the same form as Bids table:

(PetOwner username, PetName, caretaker username, date, end date, transfer type, payment type, price)

The entries for such transactions in the Bids table will be transferred to the InvalidatedBids Table, where attributes (caretaker username, date) do not reference the Availability table. Data entries would be conserved, avoiding loss of any invalidated bids placed by the PetOwners without notice to the PetOwners.

4. Description of BidsWithoutPetOwner

In real life, PetOwners may delete their accounts. Since our Bids table's primary keys includes PetOwner username and the PetName, and bids are important to calculate the salary of the caretakers, we cannot afford to have any rows deleted once a PetOwner decides to delete their account. As such, we created a parallel table named 'BidsWithoutPetOwner', which would contain the bids of deleted PetOwner accounts without referencing PetOwner and the PetName. The entries take the same form as Bids table:

(PetOwner username, PetName, caretaker username, date, end date, transfer type, payment type, price)

Since successful bidding records can now be in either the Bids or BidsWithoutPetOwner table, simply calling a function named combinedBids, we are able to query from the union of our Bids & BidsWithoutPetOwner Tables.

2.2 Constraints Not Captured by ER Model

To avoid repetition of constraints in under section "1.2 Data Requirements", we will simply list the corresponding indices of the constraints not enforced by the relational Schema. A full readable list can be found on *Annex 1.1*. They are (9a), (9b), (11a), (11b), (12), (14), (15), (15a), (16), (16a), (16b), (19a), (19bi), (19bii), (19c), (19d), (19e), (19g), (19h), (19i), (19j), (20), (20a), (20b), (20c), (20d), (20e), (20f), (21a), (21b), (21c), (21d), (21e), (22), (23), (24), (25), mostly referring to constraints requiring triggers to enforce.

3. Relational Schema

3.1 SQL Create Table Statements

```sql
CREATE TABLE Users (
    username VARCHAR PRIMARY KEY,
    email VARCHAR,
    profile VARCHAR NOT NULL,
    address VARCHAR NOT NULL,
    phoneNum INTEGER NOT NULL
);
CREATE TABLE Consumers (
    username VARCHAR PRIMARY KEY REFERENCES Users(username) ON DELETE cascade,
    creditCard INTEGER,
    bankAcc INTEGER NOT NULL
);
CREATE TABLE PCSadmins (
    username VARCHAR PRIMARY KEY REFERENCES Users(username) ON DELETE cascade
);
CREATE TABLE PetOwners (
    username VARCHAR PRIMARY KEY REFERENCES Consumers(username) ON DELETE cascade
);
CREATE TABLE CareTakers (
    username VARCHAR PRIMARY KEY REFERENCES Consumers(username) ON DELETE CASCADE,
    manager VARCHAR NOT NULL REFERENCES PCSadmins(username)
);
CREATE TABLE PartTimers (
    username VARCHAR PRIMARY KEY REFERENCES CareTakers(username) ON DELETE CASCADE
);
CREATE TABLE PetTypes (
    category VARCHAR PRIMARY KEY NOT NULL,
    baseprice FLOAT(4) NOT NULL
);
CREATE TABLE AbleToCare (
    caretaker VARCHAR REFERENCES CareTakers(username) ON DELETE CASCADE,
    category VARCHAR REFERENCES PetTypes(category),
    feeperday FLOAT(4) NOT NULL,
    PRIMARY KEY(caretaker, category)
);
CREATE TABLE Pets (
    petowner VARCHAR REFERENCES PetOwners(username) ON DELETE CASCADE,
    petname VARCHAR NOT NULL,
    profile VARCHAR NOT NULL,
    specialReq VARCHAR,
    category VARCHAR NOT NULL REFERENCES PetTypes(category),
    PRIMARY KEY (petowner, petname)
);
CREATE TABLE Availability (
    caretaker VARCHAR REFERENCES CareTakers(username) ON DELETE CASCADE,
    avail DATE NOT NULL,
    PRIMARY KEY (caretaker, avail)
);
CREATE TABLE Bids (
    petowner VARCHAR NOT NULL,
    petname VARCHAR NOT NULL,
    caretaker VARCHAR NOT NULL,
    avail DATE NOT NULL,
    edate DATE NOT NULL,
    transferType VARCHAR NOT NULL,
    paymentType VARCHAR NOT NULL CHECK(paymentType='creditcard' OR paymentType='cash'),
    price FLOAT(4) NOT NULL,
    isPaid BOOLEAN NOT NULL,
    status CHAR(1) NOT NULL CHECK(status='a' OR status='p' OR status='r'),
            /* a-accepted, p-pending, r-rejected */
    review VARCHAR,
    rating INTEGER CHECK(rating>=1 AND rating <=5),
    FOREIGN KEY(petowner, petname) REFERENCES Pets(petowner, petname) ON DELETE CASCADE,
    FOREIGN KEY(caretaker, avail) REFERENCES Availability(caretaker, avail) ON DELETE CASCADE,
    PRIMARY KEY(petowner, petname, caretaker, avail)
);
CREATE TABLE BidsWithoutPetOwner(
    petowner VARCHAR NOT NULL,
    petname VARCHAR NOT NULL,
    caretaker VARCHAR NOT NULL,
    avail DATE NOT NULL,
    edate DATE NOT NULL,
    transferType VARCHAR NOT NULL,
    paymentType VARCHAR NOT NULL CHECK(paymentType='creditcard' OR paymentType='cash'),
    price FLOAT(4) NOT NULL,
    isPaid BOOLEAN NOT NULL,
    status CHAR(1) NOT NULL CHECK(status='a'), /* a-accepted*/
    review VARCHAR,
    rating INTEGER CHECK(rating>=1 AND rating <=5),
    FOREIGN KEY(caretaker, avail) REFERENCES Availability(caretaker, avail) ON DELETE CASCADE,
    PRIMARY KEY(petowner, petname, caretaker, avail)
);
CREATE TABLE InvalidatedBids (
    petowner VARCHAR NOT NULL,
    petname VARCHAR NOT NULL,
    caretaker VARCHAR NOT NULL,
    sdate DATE NOT NULL,
    edate DATE NOT NULL,
    transferType VARCHAR NOT NULL,
    paymentType VARCHAR NOT NULL CHECK(paymentType='creditcard' OR paymentType='cash'),
    price FLOAT(4) NOT NULL,
    FOREIGN KEY(petowner, petname) REFERENCES Pets(petowner, petname) ON DELETE CASCADE,
    PRIMARY KEY(petowner, petname, caretaker, edate)
);
```

Constraints Not Enforced by Relational Schema

To avoid repetition of constraints in section "1.2 Data Requirements", we will simply list the corresponding indices of the constraints not enforced by the relational Schema. A full readable list can be found on *Annex 1.2*. They are, (2a), (2b), (3a), (10), (11a), (11b), (12), (15a), (16), (16a), (16b), (19a), (19b),(19bi), (19bii), (19c), (19d), (19e), (19g), (19h), (19i), (19j), (20a), (20b), (20c), (20d), (20e), (20f), (21a), (21b), (21c), (21d), (21e), (22), (23), (24).

Note that for constraint (2a) which is the covering constraint for Users and its subtypes, Consumers and PCS admin, we could define the schema as in the lecture notes to enforce the covering constraint, where, we keep the Consumer and PCS admin table without a User's table. However, this means that the constraint that username should be unique across all accounts is not enforced, and two separate trigger has to be created to ensure two username does not appear in either table. Instead, we retain the users table and use procedures to enforce the covering and non-overlapping constraint.

4. Normal Forms of Database

To check if the Database is in BCNF, or 3NF, we analysed each of the tables (SQL Schema), identify the functional dependencies and check if the criteria for BCNF or 3NF is fulfilled.

1) Table 'Users'
    a. Non-trivial Functional Dependencies
        i. username ->email, profile, address, phoneNum
    b. username is a superkey, so the table is in BCNF, and by implication 3NF
2) Table 'Consumers'
    a. Non-trivial Functional Dependencies
        i. username -> creditCard, bankAcc
    b. username is a superkey, so the table is in BCNF, and by implication 3NF
3) Table 'PCSadmins' has no Non-trivial Functional Dependencies, thus, it is in BCNF, and 3NF
4) Table 'PetOwners' has no Non-trivial Functional Dependencies, thus, it is in BCNF, and 3NF
5) Table 'CareTakers'
    a. Non-trivial Functional Dependencies
        i. username -> manager
    b. username is a super key, so the table is in BCNF, and by implication 3NF
6) Table 'PartTimers' has no Non-trivial Functional Dependencies, thus, it is in BCNF, and 3NF
7) Table 'PetTypes'
    a. Non-trivial Functional Dependencies
        i. category -> baseprice

    b. category is a superkey, so the table is in BCNF, and by implication 3NF

8) Table 'AbleToCare'
    a. Non-trivial Functional Dependencies
        i. caretaker, category -> feeperday
    b. caretaker, category is a superkey, so the table is in BCNF, and by implication 3NF
9) Table 'Pets'
    a. Non-trivial Functional Dependencies
        i. PetOwner, petname -> profile, specialReq, category
    b. PetOwner, petname is a superkey, so the table is in BCNF, and by implication 3NF
10) Table 'Availability' has no Non-trivial Functional Dependencies, thus, it is in BCNF, and 3NF
11) Table 'Bids'
    a. Non-trivial Functional Dependencies
        i. PetOwner, petname, caretaker, avail -> edate, transferType, paymentType, price, isPaid, status, review, rating
    b. PetOwner, petname, caretaker, avail is a superkey, so the table is in BCNF, and by implication 3NF
12) Table 'BidsWithoutPetOwner'
    a. Non-trivial Functional Dependencies
        i. PetOwner, petname, caretaker, avail -> edate, transferType, paymentType, price, isPaid, status, review, rating
    b. PetOwner, petname, caretaker, avail is a superkey, so the table is in BCNF, and by implication 3NF
13) Table 'InvalidatedBids'
    a. Non-trivial Functional Dependencies
        i. PetOwner, petname, caretaker, edate -> sdate, transferType, paymentType, price
    b. PetOwner, petname, caretaker, edate is a superkey, so the table is in BCNF, and by implication 3NF

In conclusion, with all the tables in BCNF, our database is in BCNF, and thus also 3NF.

5. Non-trivial Triggers

In this section, we look at the top three complex triggers used in our application: checkIsValidBid, checkIsValidLeave, and checkIsDeletableCareTaker. The helper functions used for some of the triggers are found in *Annex 2.0.*

5.1 Check is Valid Bid

This trigger checks if the bid is valid and raises an error and returns null if the bid is not valid. Moreover, it carries out certain procedures upon a successful update of certain values in its columns. Let's call the columns of the tuple to be inserted or updated as the 'given' column values. The constraints enforced are as follows.

Constraints enforced:

1) New bids should only be for dates greater than the current date (in the future)
2) Users who are both caretaker and PetOwners may not bid for their own services.
3) New bids should only be for dates the caretaker is available to take care of at least one more pet.
    a. Max pet is first computed for the caretaker for the specific caretaker. Then count the approved bids for the given caretaker, on the given date. If the entries are less than the max pet limit, and the date is a working day for the caretaker, then the caretaker is available to care for another pet.
4) The pet-category of a pet in the bid should match the pet-category the caretaker is able to care for in the AbleToCare table.
    a. To check for this, first obtain the category of the pet from the Pets table. Next, go to the AbleToCare table, check if the pet category found is in the category of pets column for the caretaker.
5) The bid price in bidding records should not be below the fee per day listed in the AbleToCare table by the caretaker for a particular pet-type
    a. To enforce this, use the category of the given pet we found from the previous point, then from the AbleToCare table to extract the fee per day for the caretaker for the pet's category. Lastly check if the bid price is lower than this fee per day.
6) Initial inserted bids should have NULL ratings, reviews and pending status
7) If the pet of a particular PetOwner already has an approved bid for a given date, he cannot bid for anymore services for that same pet on that same date
    a. This is enforced by filtering the combined bids table for the given PetOwner and pet, the given date, with an approved status. If this table has at least one row, then the new bid is not valid.
8) Updating a bid to approved status by caretakers should only be for bids on dates after the current date
9) Caretakers can only approve a bid if the number of pets taken for that day is within its max pet limit
    a. Similar to checking the caretaker is available to care for another pet.
10) If a bid had been rejected, it cannot be approved later.
11) When a bid is approved, all other bids pending for the pet on this same date will be rejected. Furthermore, all bids associated with the rejected bid through the same transaction is rejected.

  a. First find the columns of end dates of other pending bids with the given PetOwner, petname and avail date, but a different caretaker. Next, we reject all other pending bids with the same PetOwner, petname, as the given one, but a different caretaker from the given one, and has an end date that exists in the column of end dates previously found.

12) If a caretaker becomes fully booked (after approving a bid), all other bids pending for this caretaker on that date should be rejected. When the bid is rejected, all other bids with the same PetOwner, pet, caretaker and end date should be rejected as well.
    a. A caretaker being fully booked can be checked by counting the number of approved bids for the given caretaker on the given date, is equal to exactly one less than their max pet limit. Then we reject all other bids with the same date for this caretaker, and bids with the matching end date with the initial deleted bids. (similar to point k)
13) A rating cannot be updated on bids that are not approved, or the bid already has set a rating
14) If the update of a rating causes the average ratings of a caretaker to fall, resulting in a lower upper bound limit on the max price they can set for their feeperday to fall, check if their fee per day exceeds the upper bound, and reset it to base price if it does.
    a. To enforce this, check if the new rating changes their average rating so that the max price multiplier has changed. If it has, update the fees per day if it now exceeds the max price multiplier.
15) If the update of a rating causes the max pet limit of a PartTime caretaker to fall, then some previously available dates are no longer available, reject the bids for the unavailable dates.
    a. To enforce this, check if the new rating changes their average rating so that their max pet limit changes. If it has, the trigger checks if the caretaker is still available with the new max pet limit. If not, it deletes the pending bids for the given caretaker on the given date.
16) There can only be at most one rating and review for each transaction (continuous service provided by the same caretaker to the same pet from one date to another)
    a. This is enforced by checking that “avail”=”edate” for the bid whose rating and review is to be updated

SQL Code for Trigger (Also available in *Annex 2.1*)

```
DROP TRIGGER IF EXISTS checkIsValidBid ON Bids;
CREATE OR REPLACE FUNCTION isValidBid()
RETURNS TRIGGER AS $$
    DECLARE minfee NUMERIC;
    DECLARE petcategory VARCHAR;
    DECLARE oldmult NUMERIC;
    DECLARE newmult NUMERIC;
    DECLARE isupdate BOOLEAN;
    DECLARE oldmaxpet INTEGER;
    DECLARE newmaxpet INTEGER;
    BEGIN
        isupdate := (OLD.petowner IS NOT NULL AND OLD.petname IS NOT NULL
            AND OLD.caretaker IS NOT NULL AND OLD.avail IS NOT NULL);
        IF NOT isupdate THEN /*Insertion of a new bid*/
            IF NEW.avail<=(SELECT currentDate()) THEN
                RAISE EXCEPTION 'bids should only be for dates tomorrow
                    onwards (i.e. greater than current date)';
                RETURN NULL;
            END IF;
            IF NEW.caretaker=NEW.petowner THEN
                RAISE EXCEPTION 'a user who is both a caretaker and
                    petowner cannot bid for his own services';
                RETURN NULL;
            END IF;
            IF NOT (SELECT isAvailable(NEW.caretaker, NEW.avail)) THEN
                RAISE EXCEPTION 'caretaker is unavailable on that date';
                RETURN NULL;
            END IF;
            SELECT P.category INTO petcategory
                FROM Pets P WHERE P.petowner=NEW.petowner
                    AND P.petname=NEW.petname;
            IF petcategory NOT IN (
                    SELECT A.category
                    FROM AbleToCare A
                    WHERE A.caretaker=NEW.caretaker) THEN
                RAISE EXCEPTION 'caretaker is unable to care for this pet type';
                RETURN NULL;
            END IF;
```

```sql
            SELECT A.feeperday INTO minfee
                FROM AbleToCare A WHERE A.caretaker=NEW.caretaker
                    AND A.category=petcategory;
            IF NEW.price<minfee THEN
                RAISE EXCEPTION 'bid price is below caretakers fee per day for that pet';
                RETURN NULL;
            END IF;
            IF NEW.status!='p' OR NEW.rating IS NOT NULL
                    OR NEW.review IS NOT NULL THEN
                RAISE EXCEPTION 'initial status and rating for bid should be pending and null';
                RETURN NULL;
            END IF;
            IF EXISTS (
                    SELECT *
                    FROM combinedBids() B
                    WHERE B.petowner=NEW.petowner AND B.petname=NEW.petname
                        AND B.avail=NEW.avail AND B.status='a') THEN
                RAISE EXCEPTION 'pet already has an existing appointment
                    with another caretaker on this date';
                RETURN NULL;
            END IF;
            RETURN NEW;
        END IF;
        /*Is an update*/
        IF NEW.status!=OLD.status THEN /*status update*/
        /*When updating to approve bid*/
            IF NEW.status='a' THEN
                IF NEW.avail<=(SELECT currentDate()) THEN
                    RAISE EXCEPTION 'approved bids should only be for dates
                            tomorrow onwards (i.e. greater than current date)';
                    RETURN NULL;
                END IF;
                /*When updating to approve a bid, check if number of pets
                        taken for that day is within max pet limit*/
                IF NOT (SELECT isAvailable(NEW.caretaker, NEW.avail)) THEN
                    RAISE EXCEPTION 'this date is fully booked, cannot approve';
                    RETURN NULL;
                END IF;
                IF OLD.status='r' THEN /*check the bid is not already rejected*/
                    RAISE EXCEPTION 'cannot approve already rejected bid';
                    RETURN NULL;
                END IF;
                /*When updating to approve bid, all other bids pending for
                        the pet on this same date will be rejected*/
                /*Note when we reject a bid on an avail date, all other
                        similar bids with same edate should also be rejected*/
                UPDATE Bids SET status='r'
                    WHERE petowner=NEW.petowner AND petname=NEW.petname
                        AND caretaker!=NEW.caretaker AND status='p'
                        AND NEW.edate IN
                            (SELECT B.edate
                                FROM Bids B
                                WHERE B.petowner=NEW.petowner
                                    AND B.petname=NEW.petname AND B.status='p'
                                    AND B.avail=NEW.avail);
                /*If caretaker becomes fully booked, all other bids pending
                        for this caretaker on this date will be rejected*/
                /*Note when we reject a bid on an avail date, all other
                        similar bids with same edate should also be rejected*/
                IF (SELECT COUNT(*)
                        FROM combinedBids() B
                        WHERE B.caretaker=NEW.caretaker
                            AND B.avail=NEW.avail AND B.status='a'
                        )=(SELECT computeMaxPet(NEW.caretaker)-1) THEN
                    UPDATE Bids SET status='r'
                        WHERE caretaker=NEW.caretaker AND edate!=NEW.edate
                            AND status='p'
                            AND edate IN
```

```
                                    (SELECT B.edate
                                        FROM Bids B
                                        WHERE B.caretaker=NEW.caretaker
                                            AND B.avail=NEW.avail AND B.status='p');
                END IF;
            END IF;
        END IF;
        /*Update on rating*/
        IF NEW.rating!=OLD.rating
                OR (NEW.rating IS NOT NULL AND OLD.rating IS NULL)
                OR (NEW.rating IS NULL AND OLD.rating IS NOT NULL) THEN
            IF OLD.avail!=OLD.edate THEN
                RAISE EXCEPTION 'Can only rated once for each transaction,the availability
                        should equal end date for the bid to be rated';
                RETURN NULL;
            END IF;
            IF NEW.rating IS NOT NULL THEN
                IF OLD.status!='a' THEN
                    RAISE EXCEPTION 'cannot update ratings on bid that is
                            not approved';
                    RETURN NULL;
                END IF;
                IF OLD.rating IS NOT NULL THEN
                    RAISE EXCEPTION 'cannot update rating that has been set already';
                    RETURN NULL;
                END IF;
                /*Update fees per day if average ratings change*/
                SELECT computeMaxPriceMultiplier(OLD.caretaker) INTO oldmult;
                SELECT computeUpdatedMaxPriceMultiplier(OLD.caretaker, NEW.rating)
                        INTO newmult;
                IF newmult!=oldmult THEN
                    UPDATE AbleToCare SET feeperday=(SELECT getBasePrice(category))
                        WHERE caretaker=OLD.caretaker
                            AND feeperday>newmult*(SELECT getBasePrice(category));
                END IF;
                IF OLD.caretaker IN (SELECT P.username FROM PartTimers P) THEN
                    /*Only part-timers*/
                    SELECT computeMaxPet(OLD.caretaker) INTO oldmaxpet;
                    SELECT computeUpdatedMaxPet(OLD.caretaker, NEW.rating)
                            INTO newmaxpet;
                    /*if max pets decrease, some previously available dates no longer available
                     *  then have to update pending bids on unavailable dates*/
                    IF newmaxpet<oldmaxpet THEN
                        UPDATE Bids SET status='r'
                            WHERE caretaker=OLD.caretaker AND status='p'
                                AND edate IN
                                (SELECT B.edate
                                    FROM Bids B
                                    WHERE B.caretaker=caretaker AND B.status='p'
                                        AND NOT (SELECT isAvailable(caretaker, avail, newmaxpet)) );
                    END IF;
                END IF;
            ELSE /*new rating is null and old rating is not null*/
                RAISE EXCEPTION 'cannot update rating that has been set already to null';
                RETURN NULL;
            END IF;
        END IF;
        /*update on review*/
        IF NEW.review!=OLD.review OR (NEW.review IS NOT NULL AND OLD.review IS NULL)
                OR (NEW.review IS NULL AND OLD.review IS NOT NULL) THEN
            IF OLD.avail!=OLD.edate THEN
                RAISE EXCEPTION 'Can only review once for each transaction,
                        the availability should equal end date for the bid to be reviewed';
                RETURN NULL;
            END IF;
        END IF;
        RETURN NEW;
    END; $$
```

```
LANGUAGE plpgsql;
CREATE TRIGGER checkIsValidBid
BEFORE INSERT OR UPDATE ON Bids
FOR EACH ROW EXECUTE PROCEDURE isValidBid();
```

5.2 Check is Valid Leave

This triggers checks that a caretaker applying for leave (deleting dates from the Availability table) are FullTimers who meet the consecutive working days condition. However, it first checks that the caretaker exists in CareTakers table. This condition will be true when cascading a deletion of the caretaker's account. When that happens, it simply returns OLD, allowing the cascading deletion to bypass the conditions of leave application. That aside, the constraints enforced on Availability is as given. The constraints enforced are as follows.

Constraints enforced

1) PartTimers cannot take leave, that is delete availability dates from the Availability table. FullTimers can take leave, but must not have any approved bids for that date
2) FullTimers can take leave on a particular date if their availability dates remaining in the Availability table fulfils the minimum 2x150 consecutive days of working days.
    a. First filter out the availability dates for the particular caretaker less the availability date to be deleted, call this column 1
    b. Next extract from column 1, all available dates where the date of the available date minus one day is not in column 1, call this column 2. These are the lower bound dates of their working day date ranges.
    c. Next extract from column 1, all available dates where the date of the available date plus one day is not in column 1, call this column 2. These are the upper bound dates.
    d. Inner join column 1 and column 2 on column 1 date ≤ column 2 date to form multiple possible date ranges (from column 1 date to column 2 date)
    e. Group by column 1 date and retain the rows with the smallest difference between the two date as the actual start and end date of the date range of the consecutive working days of the caretaker. This works because the date ranges cannot overlap. Thus, we have a table of the date ranges of the working days of the worker.
    f. If the number of date ranges that span over 150 days is greater or equal to two, or the maximum date range span is above and equal to 300, then the worker fulfills the 2x150 workdays condition and the leave is valid.
3) When leave is successfully applied, pending or rejected bids on the leave dates should have its data transferred to the Invalidated Bids table before being deleted.
    a. First select from Bids the row with the given PetOwner, PetName, caretaker and availability date, and find the corresponding end date, next select all the Bids with the given PetOwner, PetName, caretaker and the end date found. Group by PetOwner, PetName, caretaker and end date, then pick the minimum availability date as the start date.
    b. Natural join with Bids table to get the values of the other remaining relevant fields for Invalidated Bids table, and add those rows to the table if it does not already exist in the table.

SQL Code for Trigger (Also available in *Annex 2.2*)

```
DROP TRIGGER IF EXISTS checkIsValidLeave ON Availability;
CREATE OR REPLACE FUNCTION isValidLeave()
RETURNS TRIGGER AS $$
    DECLARE yearofleave INTEGER;
    DECLARE isvalid150days BOOLEAN;
    DECLARE error VARCHAR;
    BEGIN
        IF OLD.caretaker NOT IN (SELECT username FROM caretakers) THEN
                /*This means delete cascade from caretakers is called*/
            RETURN OLD; /*bypass all the validLeave conditions*/
        END IF; /*This condition to ensure on delete cascade from caretakers still work*/
        IF OLD.caretaker IN (SELECT P.username FROM PartTimers P) THEN
            RAISE EXCEPTION 'part-timers cannot take leave';
            RETURN NULL;
        END IF;
        IF OLD.avail IN (SELECT B.avail FROM combinedBids() B
                            WHERE B.caretaker=OLD.caretaker AND B.status='a') THEN
            RAISE EXCEPTION 'caretaker is occupied that day';
            RETURN NULL;
        END IF;
        yearofleave := (SELECT EXTRACT(YEAR FROM OLD.avail));
        isvalid150days := (
                        SELECT COUNT(*)>=2
                        OR (COUNT(*)>=1 AND MAX(D.enddate-D.startdate+1)>=300)
                    FROM
                        (SELECT X.avail AS startdate, X.avail+MIN(Y.avail-X.avail) AS enddate
                        FROM
                            (SELECT AV.avail
                            FROM Availability AV
                            WHERE AV.caretaker=OLD.caretaker AND AV.avail!=CAST(OLD.avail AS DATE)
                                AND (SELECT EXTRACT(YEAR FROM AV.avail))=yearofleave
                                AND AV.avail-1 NOT IN (SELECT A.avail
                                                        FROM Availability A
                                                        WHERE A.caretaker=OLD.caretaker
                                                        AND A.avail!=CAST(OLD.avail AS DATE)
                                                        AND (SELECT EXTRACT(YEAR FROM A.avail))
                                                                =yearofleave) ) X
                            INNER JOIN
                            (SELECT AV.avail
                            FROM Availability AV
                            WHERE AV.caretaker=OLD.caretaker AND AV.avail!=CAST(OLD.avail AS DATE)
                                AND (SELECT EXTRACT(YEAR FROM AV.avail))=yearofleave
                                AND AV.avail+1 NOT IN (SELECT A.avail
                                                        FROM Availability A
                                                        WHERE A.caretaker=OLD.caretaker
                                                        AND A.avail!=CAST(OLD.avail AS DATE)
                                                        AND (SELECT EXTRACT(YEAR FROM A.avail))
                                                                =yearofleave) ) Y
                        ON X.avail<= Y.avail
                        GROUP BY X.avail ) D
                    WHERE D.enddate-D.startdate+1>=150 );
        IF NOT isvalid150days THEN
            RAISE EXCEPTION 'violates 150 days constraint';
            RETURN NULL;
        END IF;
        /*place pending or rejected bids affected by leave into invalidated bids*/
        INSERT INTO InvalidatedBids(petowner, petname, caretaker, sdate, edate, transferType, paymentType, price)

        SELECT X.petowner, X.petname, X.caretaker, X.avail, X.edate, Y.transferType, Y.paymentType, Y.price
        FROM
            (SELECT D.petowner, D.petname, D.caretaker, MIN(D.avail) AS avail, D.edate
            FROM Bids D
            WHERE caretaker=OLD.caretaker AND status!='a'
                AND edate IN
                (SELECT B.edate
                FROM Bids B
                WHERE B.caretaker=caretaker AND B.status!='a' AND B.avail=OLD.avail )
            GROUP BY D.petowner, D.petname, D.caretaker, D.edate) X
            NATURAL JOIN Bids Y
        WHERE NOT EXISTS (SELECT *
                            FROM InvalidatedBids I
                            WHERE I.petowner=X.petowner AND I.petname=X.petname
                            AND I.caretaker=X.caretaker AND I.edate = X.edate)
        ;/*delete bids on leave dates*/
        DELETE FROM Bids
            WHERE caretaker=OLD.caretaker AND status='p'
                AND edate IN
                (SELECT B.edate
                FROM Bids B
                WHERE B.caretaker=caretaker AND B.status='p' AND B.avail=OLD.avail );
        RETURN OLD;
    END; $$
LANGUAGE plpgsql;
CREATE TRIGGER checkIsValidLeave
BEFORE DELETE ON Availability
FOR EACH ROW EXECUTE PROCEDURE isValidLeave();
```

5.3 Check is Deletable CareTaker

This triggers checks that a CareTaker account can be deleted so long as the CareTaker does not have any more approved bids yet to be serviced in the future. If the CareTaker account is deleted, carry out a procedure to store affected transactions in Bids into the Invalidated Bids table. The constraints enforced are as follows.

Constraints enforced

1) A caretaker account cannot be deleted if there are still approved bids in the future (after the current date)
2) Once a caretaker is deleted, the respective rows in the Bids table for that caretaker will have the information converted and stored in the Invalidated Bids table, and then subsequently deleted.
    a. Note that for this, procedure, we cannot rely on the trigger for the deletion of caretakers Availability upon "on delete cascade" to store the information of deleted Bids into the Invalidated Bids table because for the cascade deletion to work, the criteria on the deletion triggers for Availability table has to be bypassed completely.
    b. The implementation process to convert and store values in invalidated Bids is the same as in part (3) of the Check is Valid Leave trigger.

SQL Code for Trigger (Also available in *Annex 2.3*)

```sql
DROP TRIGGER IF EXISTS checkIsDeletableCareTaker ON CareTakers;
CREATE OR REPLACE FUNCTION isDeletableCareTaker()
RETURNS TRIGGER AS $$
    BEGIN
        IF EXISTS (SELECT * FROM combinedBids() B
                WHERE B.caretaker=OLD.username AND B.status='a' AND (SELECT currentDate())<=B.avail) THEN
            RAISE EXCEPTION 'this caretaker is servicing a petowner in the future!';
            RETURN NULL;
        END IF;
        /*place pending or rejected bids affected by removal of caretaker into invalidated bids*/
        INSERT INTO InvalidatedBids(petowner, petname, caretaker, sdate, edate, transferType, paymentType, price)

        SELECT X.petowner, X.petname, X.caretaker, X.avail, X.edate, Y.transferType, Y.paymentType, Y.price
        FROM
            (SELECT D.petowner, D.petname, D.caretaker, MIN(D.avail) AS avail, D.edate
            FROM Bids D
            WHERE caretaker=OLD.username AND status!='a'
            GROUP BY D.petowner, D.petname, D.caretaker, D.edate) X
            NATURAL JOIN Bids Y
        WHERE NOT EXISTS (SELECT *
                        FROM InvalidatedBids I
                        WHERE I.petowner=X.petowner AND I.petname=X.petname
                        AND I.caretaker=X.caretaker AND I.edate = X.edate);
        RETURN OLD;
    END; $$
LANGUAGE plpgsql;
CREATE TRIGGER checkIsDeletableCareTaker
BEFORE DELETE ON CareTakers
FOR EACH ROW EXECUTE PROCEDURE isDeletableCareTaker();
```

6. Interesting Queries

6.1 Finding Available CareTakers

This query is meant for PetOwners to browse and select caretakers that are available to care for their pets. A caretaker is considered available for bidding, if he has a schedule to work on all the days specified, has daily pet slots left for caretaking service and can take care of that pet category. We also show the average rating so that PetOwners can choose caretakers with a good track record.

1. First, we have a query to compute the max pet limit for each caretaker, based on whether they are full-time, or if they are parttime, based on their rating. Next, using the Bids table, we tabulate the number of pets each caretaker has to care for on each of their work days using a group by clause on caretaker and their workday (avail). After this, we filter out only the workdays where the number of pets they have to care for is strictly less than their max pet limit. By this time, we have a table of the caretaker's and dates they can care for one more pet, which we then restrict to the dates within the input date range.
2. Finally, to check if the caretaker is available to care for another pet for each and every day within the stipulated date range, we count the number of available dates remaining for each caretaker and compare it to the number of days in the date range. If the count is the same, then they must be available everyday within this date range, thus we extract these caretakers out. With these set of available caretakers, we filter out the caretakers are not able to care for the pet category desired by the pet owner. With these remaining caretakers, we compute their average ratings, number of ratings, and their

fee per day for the pet category of interest, and join them to the respective caretakers. Thus, the petowner will be able to browse available caretakers for their pet, as well as the ratings and prices of these caretakers.
Utilizing cases to compute the max pet limit of each caretaker
Group-by: Having avail dates from PetOwner defined start - end dates

SQL Code for Query (Also available in *Annex 2.4*)

```sql
CREATE OR REPLACE FUNCTION
        viewAvailability(startdatein DATE, enddatein DATE, petcategory VARCHAR)
RETURNS TABLE(
    caretaker VARCHAR, category VARCHAR, feeperday FLOAT(4), avgrating NUMERIC,
    numratings BIGINT, startdate DATE,enddate DATE) AS $$
    BEGIN
        RETURN QUERY (
            SELECT atc.caretaker, atc.category, atc.feeperday,
                AVGRC.avgrating, AVGRC.numratings,
                startdatein AS startdate, enddatein AS enddate
            FROM AbleToCare atc
                NATURAL JOIN ( /*Table of each caretaker's average rating and number of rating*/
                    SELECT CB.caretaker, AVG(CB.rating) AS avgrating,
                        COUNT(CB.rating) AS numratings
                    FROM (SELECT * FROM Bids UNION
                            SELECT * FROM BidsWithoutPetOwner) CB
                    GROUP BY CB.caretaker
                    UNION
                    SELECT CT.username AS caretaker,
                        NULL AS avgrating, 0 AS numratings
                    FROM CareTakers CT
                    WHERE CT.username NOT IN (
                        SELECT CB1.caretaker
                        FROM (SELECT * FROM Bids UNION
                                SELECT * FROM BidsWithoutPetOwner) CB1
                        )
                    ) AVGRC
            WHERE EXISTS
                (SELECT 1 FROM
                    (SELECT availCT.caretaker, COUNT(*) AS days
                        FROM /*table of caretaker's availability to take on another
                                pet on that date, within a restricted date range*/
                            (SELECT AV.caretaker, AV.avail
                                FROM /*table of tabulating number of pets a caretaker
                                    has to take care on a work day*/
                                    (SELECT B.caretaker, B.avail, COUNT(*) AS cnt
                                        FROM Bids B
                                        WHERE B.status ='a'
                                        GROUP BY B.caretaker, B.avail
                                        UNION
                                        SELECT A.caretaker, A.avail, 0 AS cnt
                                        FROM Availability A
                                        WHERE NOT EXISTS (
                                            SELECT *
                                            FROM Bids B
                                            WHERE B.caretaker=A.caretaker
                                                AND B.avail=A.avail AND B.status='a'
                                            )
                                    ) AV
                                WHERE AV.cnt<(/*This nested query computes
                                            the max pet limit of AV.caretaker*/
                                                SELECT CASE
                                                    WHEN AV.caretaker NOT IN (
                                                        SELECT P.username FROM PartTimers P
                                                        ) THEN 5
                                                        /*means caretaker is full-time,
                                                        default value 5.
                                                        Look at part-time case below*/
                                                    WHEN AVGR.avgrating>4.7 THEN 5
                                                    WHEN AVGR.avgrating>4.0 THEN 4
                                                    WHEN AVGR.avgrating>3.0 THEN 3
```

```
                                                       ELSE 2
                                                       END
                                                   FROM (SELECT AVG(CB.rating) AS avgrating
                                                           FROM (SELECT * FROM Bids UNION
                                                               SELECT * FROM BidsWithoutPetOwner) CB
                                                           WHERE CB.caretaker=AV.caretaker) AVGR
                                                   )
                                      AND (AV.avail BETWEEN startdatein AND enddatein)
                                      /*restrict to date range*/
                               )AS availCT
                      GROUP BY availCT.caretaker
                      HAVING COUNT(*) = (enddatein-startdatein) +1
                      /*means caretaker is available evey day within the date range*/
                      ) AS t
                  WHERE t.caretaker = ATC.caretaker
                  )
                  AND atc.category = petcategory
              ORDER BY avgrating DESC NULLS LAST, numratings DESC, feeperday ASC
        );
    END;$$
LANGUAGE plpgsql;
```

6.2 Calculating Monthly Salaries for CareTakers

This query is meant for the PCS admin to see the wages of the caretaker, as well as identify any weak caretakers in terms of their ratings or number of pet days clocked for a particular month. A caretaker's monthly pay consists of approved bids, that are paid and serviced within that month itself. FullTimers and PartTimers each have their own wage calculation.

1. The computation of wages and number of pet days comprise of computing those values for part-timers and full-timers separately, and then union the results to form a combined table for all caretakers. For the part-time caretakers, we identify the approved and paid bids for each care-taker within the month, and then group those bids by caretakers. Next we sum the prices of the bids within each group and multiply it by 0.75, and assign those values as wages to the part-timers. To get the pet days clocked, we simply count the rows in each group. Part-timers who have no bids that meet the approved and paid criteria are assigned zero salary and pet days.
2. For the full timers, we iterate for each full-timer the following procedure. For each full-timer, we identify all their approved, and paid bids in the month, sort them in ascending order and cut off the first 60 bids. The remaining bids are the excess pet days, where we can sum their prices and multiply by 0.8. Finally, we add their base salary of 3000 dollars to get their full salary. We also count the number of approved paid bids for a caretaker in a month as their pet days clocked.
3. We union the part-time and full-timers to get a combined caretaker's wage table. Finally, we compute the average rating and assign them to the respective caretakers using a join, then we order in increasing order, the result by ratings, then pet days clocked. Thus the PCSadmin can identify the weakest caretakers as well observe their wages for that month.
   Advantages: We can easily split the query up to just look at full-time or part-time caretakers' wages.

<u>SQL Code for Query</u> (Also available in *Annex 2.5*)

```
CREATE OR REPLACE FUNCTION viewCareTakersWagePetDaysRatings(datein DATE)
RETURNS TABLE(
    caretaker VARCHAR, contract TEXT, salary FLOAT,
    petdaysclocked BIGINT, avgrating NUMERIC, numratings BIGINT) AS $$
    BEGIN
        RETURN QUERY (
            SELECT WG.caretaker, WG.contract, WG.salary,
                    WG.petdaysclocked, RT.avgrating, RT.numratings
            FROM
                (/*This table tabulates the salary and pet days clocked
                    for part-timers with at least 1 approved paid bid*/
                SELECT CB.caretaker, 'Part-Time' AS contract,
                        SUM(price)*0.75 AS salary, COUNT(*) AS petdaysclocked
                FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
                WHERE CB.avail <= CAST(date_trunc('month', datein)
                        + interval '1 month - 1 day' AS DATE)
                    AND CB.avail >= CAST(date_trunc('month', datein) AS DATE)
                    AND CB.isPaid = TRUE
                    AND CB.status = 'a'
    AND CB.caretaker IN (SELECT PT.username FROM PartTimers PT)
  GROUP BY CB.caretaker
  UNION
  /*This table tabulates the salary and pet days
    clocked for part-timers with no approved paid bid*/
  SELECT PT.username AS caretaker, 'Part-Time' AS contract,
      0.0 AS salary , 0 AS petdaysclocked
  FROM PartTimers PT
  WHERE PT.username NOT IN (
      SELECT B.caretaker
      FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) B
      WHERE B.avail <= CAST(date_trunc('month', datein)
          + interval '1 month - 1 day' AS DATE)
        AND B.avail >= CAST(date_trunc('month', datein) AS DATE)
        AND B.isPaid = TRUE
        AND B.status = 'a')
  UNION
  /*This table tabulates the salary and pet days clocked for
    full-timers with at least 1 approved paid bid*/
  SELECT CT.username AS caretaker, 'Full-Time' AS contract, (
      /*computation of salary for one full-timer*/
      SELECT CASE
          WHEN SUM(OFS.price) IS NOT NULL
            THEN 3000+SUM(OFS.price)*0.8
          ELSE 3000
          END
      FROM ( /*Find prices of all excess pet days*/
        SELECT *
        FROM (SELECT * FROM Bids UNION
            SELECT * FROM BidsWithoutPetOwner) CB
        WHERE CB.caretaker=CT.username
          AND CB.avail <= CAST(date_trunc('month', datein)
              + interval '1 month - 1 day' AS DATE)
          AND CB.avail >= CAST(date_trunc('month', datein) AS DATE)
          AND CB.isPaid = TRUE
          AND CB.status = 'a'
        ORDER BY CB.price ASC
        OFFSET 60  /*First 60 entries are assigned to the base pet days*/
        ) OFS
    ) AS salary,
    ( /*counting the pet days clocked for this full-timer*/
    SELECT COUNT(*) AS petdaysclocked
    FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
    WHERE CB.caretaker=CT.username
      AND CB.avail <= CAST(date_trunc('month', datein)
          + interval '1 month - 1 day' AS DATE)
      AND CB.avail >= CAST(date_trunc('month', datein) AS DATE)
      AND CB.isPaid = TRUE
      AND CB.status = 'a'
    ) AS petdaysclocked
  FROM CareTakers CT
  WHERE CT.username NOT IN (SELECT PT.username FROM PartTimers PT)
    /*condition to select only full-timers*/
  ) WG
  NATURAL JOIN (
  /*This table computes the average ratings for each caretaker*/
  SELECT CB.caretaker, AVG(CB.rating) AS avgrating,
      COUNT(CB.rating) AS numratings
  FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
  GROUP BY CB.caretaker
  UNION
  SELECT CT.username AS caretaker, NULL AS avgrating, 0 AS numratings
  FROM CareTakers CT
  WHERE CT.username NOT IN (
      SELECT CB1.caretaker
      FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB1)
  ) RT
ORDER BY RT.avgrating ASC, RT.numratings ASC, WG.petdaysclocked ASC
```

```
            /*order by average ratings, number of ratings, then pet days clocked*/
        );
    END;$$
LANGUAGE plpgsql;
```

6.3 Monthly Business Financials of PCS

This query allows the PCS admin to view their business financials; the total profit & revenue for hosting online services, paid salaries to all employees and pets served as a monthly time series data (over a restricted range of months).

1. Using Common Table Expressions (CTE), we create a table of all approved and successfully paid bids for a restricted range of month as defined by the admin. They can be found in both the Bids & BidsWithoutPetOwner tables. Through this CTE, all approved and paid services from a caretaker for the PetOwner and their pets, with the corresponding paid price and caretaking days are returned, for each bid. Furthermore, the month and year of each of these bids are computes in two separate columns. This CTE would be referred to as *transactions*. Next, we split the problem into two, we first calculate the total salary paid for each month in one table, and the revenue as well as number of pets served each month in another table.
2. For the computation of salary, we adopt the same methods in section (6.2), for calculating the salary of a part-timer versus a full-timer. However, in this case, we do it together in one SQL query rather than splitting it into two queries that are union together. For each caretaker in the database, we do a case analysis. If the caretaker is a part-timer (in the PartTimers table), then we compute the wage based on 0.75 times the sum of their approved and paid bid prices for the month using the transactions table. Otherwise, he must be a full-timer, and we compute the salary for the excess pet days plus the base salary of 3000 dollars for each month, similar to the previous query in section (6.2), but this time on the transactions table. With the monthly salary of each caretaker computed, to get the total salary paid for each month, we simply group by the month and sum all the caretaker's salary within each month.
3. To get the total revenue, we group the transaction table in terms of months, and sum the bid prices within the group to get the monthly revenue. We can also count the number of unique pets in the group to find the total pets serviced for that month.
4. Finally, we natural join the two tables on the month and year columns to get the combined result, with profit for the month computed as revenue minus cost of paying salary for each month. Here, the PCS admin can see the profit, revenue, salary paid as a cost, and the number of pets services evolving over the months.

   Advantages: Fast bookkeeping records for admins, skipping manual calculations and observe application's performance

   Group-by: Year and Month specified by Admin

<u>SQL Code for Query</u> (Also available in *Annex 2.6*)

```
CREATE OR REPLACE FUNCTION viewMonthlyFinancials(startdatein DATE, enddatein DATE)
RETURNS TABLE(
    year_ FLOAT, month_ FLOAT, profit FLOAT,
    totalrevenue REAL, totalsalarypaid FLOAT, totalpets BIGINT ) AS $$
    BEGIN
        RETURN QUERY (
            WITH transactions AS (
                SELECT CB.petowner, CB.petname, CB.caretaker,
                    CB.avail, CB.price,
                    EXTRACT(MONTH FROM CB.avail) AS month_,
                    EXTRACT(YEAR FROM CB.avail) AS year_
                FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
                WHERE CB.status='a' AND CB.isPaid
                    AND CB.avail <=CAST(date_trunc('month', enddatein)
                            + interval '1 month - 1 day' AS DATE)
                    AND CB.avail >=CAST(date_trunc('month', startdatein) AS DATE)
                )
            SELECT TSP.year_, TSP.month_,
                TRP.totalrevenue-TSP.totalsalarypaid AS profit,
                TRP.totalrevenue, TSP.totalsalarypaid, TRP.totalpets
            FROM (/*This table tabulates the total salary paid for each month*/
                SELECT CTS.year_, CTS.month_,
                    SUM(CTS.salary) AS totalsalarypaid
                FROM ( /*This table tabulates the salary for each caretaker
                        for a given month and year*/
                    SELECT CT.username AS caretaker, MY.year_, MY.month_,
                        (SELECT CASE /*compute salary for each caretaker*/
                            /*Full-timers case*/
                            WHEN CT.username NOT IN
                                (SELECT PT.username FROM PartTimers PT)
                            THEN
                            (SELECT CASE
                                WHEN SUM(OFS.price) IS NOT NULL
                                    THEN 3000+SUM(OFS.price)*0.8
                                ELSE 3000
                                END
                            FROM (
                                SELECT *
                                FROM transactions CB
                                WHERE CB.caretaker=CT.username AND CB.year_=MY.year_
                                    AND CB.month_=MY.month_
                                ORDER BY CB.price ASC
                                OFFSET 60
                                ) OFS
                            )
                        ELSE /*part-timers case*/
                            (SELECT CASE
                                WHEN SUM(price) IS NOT NULL THEN SUM(price)*0.75
                                ELSE 0
                                END
                            FROM transactions CB
                            WHERE CB.caretaker=CT.username AND CB.year_=MY.year_
                                AND CB.month_=MY.month_
                            )
                        END
                        )AS salary
                    FROM CareTakers CT,
                        (SELECT DISTINCT T0.month_,
                            T0.year_ FROM transactions T0) MY
                    ) AS CTS
                GROUP BY CTS.year_, CTS.month_
                ) TSP
                NATURAL JOIN
                ( /*This table tabulates the total revenue and the
                    total number pets served for each month*/
                SELECT TR0.year_, TR0.month_, SUM(TR0.price) AS totalrevenue,
                        COUNT(DISTINCT CONCAT(TR0.petowner, ',', TR0.petname) ) AS totalpets
                FROM transactions TR0
                GROUP BY TR0.year_, TR0.month_
                ) TRP
            ORDER BY TSP.year_ ASC, TSP.month_ ASC
        );
    END;$$
LANGUAGE plpgsql;
```

7. Software Tools and Frameworks

Our project has been implemented based on the following client-server model:

7.1 Server:

Our server is deployed on Heroku at https://shielded-oasis-35437.herokuapp.com/

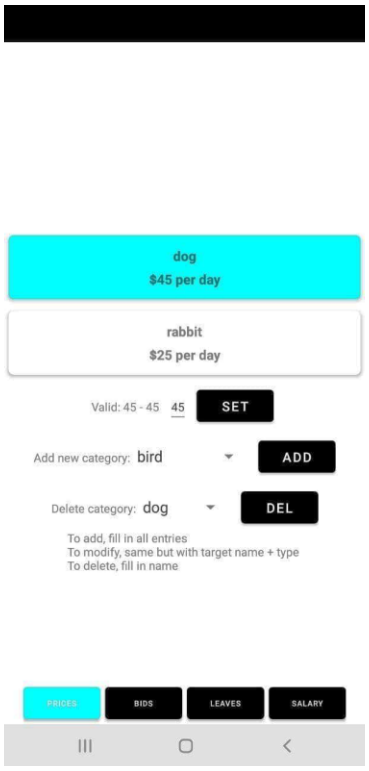
- Stack:
    - PostgreSQL Version 12.4, via heroku-postgresql service addon.
    - Express Version 4.17.1
    - Node.js Version 12.18.3

7.2 Client:

We have developed an Android app as our client application. The application follows the Model-View-ViewModel (MVVM) architecture, using the Java Retrofit client.

8. Representative Screenshots

[redacted]



dog
$45 per day

rabbit
$25 per day

Valid: 45 - 45 45

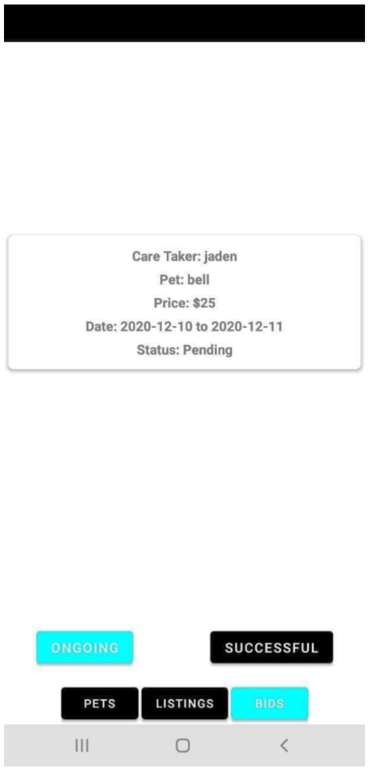
Add new category: bird

Delete category: dog

To add, fill in all entries
To modify, same but with target name + type
To delete, fill in name

Caretaker setting pet types that he can take care of, and their base price.

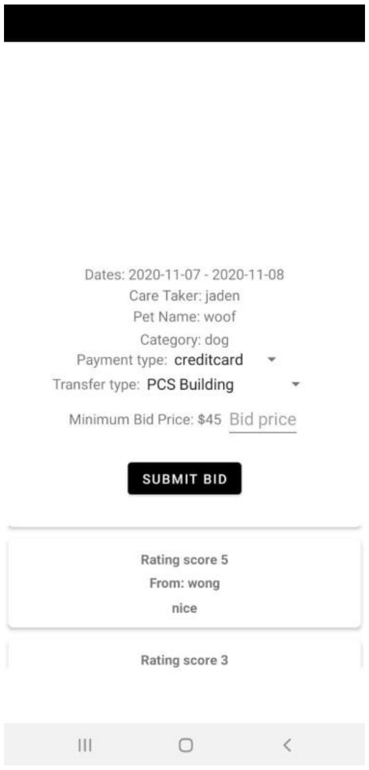
[redacted]



Care Taker: jaden
Pet: bell
Price: $25
Date: 2020-12-10 to 2020-12-11
Status: Pending

PetOwner viewing his ongoing bids.

[redacted]



Dates: 2020-11-07 - 2020-11-08
Care Taker: jaden
Pet Name: woof
Category: dog
Payment type: creditcard
Transfer type: PCS Building
Minimum Bid Price: $45 Bid price

Rating score 5
From: wong
nice

Rating score 3

PetOwner entering Bid and viewing past reviews by previous customers.

9. Difficulties Faced and Lessons Learnt

As many queries were complex, sometimes we faced difficulties on debugging our code. We learnt to debug a portion of the query at a time and create test cases that allow us to visualize whether our code is working.

As we consider more corner cases (such as deletion of caretaker or caretaker take leave when there are pending bids), we encountered some instances where we had to refine the design of the database to capture them. A lot of discussions were done to improve the database design, and from the discussions, we sharpened our skills in designing a better database.

<u>Annex 1.1</u>

<u>Constraints Not Captured by Entity-Relationship Diagram</u>

9) Full Timer and PartTimer.
    a. All Full Timers have a max pet limit of 5.
    b. PartTimers have a max pet limit between 2 to 5 which is dependent on the average rating of the PartTimer. If the average rating of the PartTimer is above 3 but below or equals to 4, limit is set to 3. If the average rating is above 4 and below or equals to 4.7, the limit is set to 4. If the rating is above 4.7, the limit is set to 5. If the rating of the PartTimer drops, resulting in a decrease in limit, successful bookings that exceeds the new limit will be kept. However, all future bids and approval of bids will be subjected to the limit.

11) CareTaker's Fee per Day
    a. This fee is greater than or equals the base price of the category of Pet Type.
    b. Fee per Day can be priced up to 1.1 times of base price if the CareTaker has an averaged rating of 4 or higher, and up to 1.2 times of base price if the CareTaker has an average rating of 4.7 or higher as the upper limit, otherwise the upper limit is equal to the base price.

12) When the PCS Admin updates the base price for a particular pet category, check if the existing fee per day for any caretaker for that category exceeds the upper limit, or falls below the base price. If so, reset to new base price.

14) FullTime CareTakers are assumed to be available every day for the whole of the next two years, unless they decide to take leave on a particular Date.

15) Part-Time CareTakers have to update their available dates for the next two years.
    a. They also cannot apply for leave.

16) FullTimers can apply for Leave Days to be taken. These Leave days allow their Availability dates (working days) to be removed from consideration, their Available dates for the year will be tracked (their leave dates are simply dates not in Available dates). There are two conditions that have to be satisfied for the new Leave to be approved.
    a. They have no Pet booked to care for during that date.
    b. Their leave should not violate the minimum 2x150 consecutive days of work for the year.

19) PetOwners bidding for CareTaker's services on behalf of their Pet
    a. PetOwners can only bid for available dates in the future (after current date)
    b. A PetOwner may bid for a range of consecutive dates for the CareTaker.
        i. This will create individual bid entries for each of the consecutive Available dates bidded for, with the same values for all the other attributes for the bidding record.
        ii. These multiple bids for consecutive dates should all have the same End Date as an indication that the bid for these multiple bids across a range of dates should be considered together as a single transaction.
    c. A CareTaker's services are only open for bidding for a new bid, if they are Available(working) on a particular day and have at least 1 remaining pet slot.
    d. The PetOwner should also bid only for CareTakers that can care for their Pet's Type for their bid to be valid.
    e. If the Pet of a particular PetOwner already has an approved bid for a given date, he/she cannot bid for anymore services for that same pet on that same date.
    g. Every bid must have a bid price greater than or equal to the CareTaker's fee per day for that pet category.
    h. Each bid will appear in the Bidding Records with a pending status initially.
    i. Each bid will initially have null values for review and rating.
    j. Users who are both caretaker and PetOwners may not bid for their own services.

20) CareTakers will review the bids and accept the bids according to his preference.
    a. Updating a bid to approved status by the CareTaker should only be for bids on dates after the current date.
    b. A bid can only be approved by the CareTaker if he/she is available to take at least one more pet without exceeding his/her max pet limit.
    c. A CareTaker can only reject or accept all the bids from a Pet and PetOwner within a single transaction as a group, that is, the set of bids with a common end date.
    d. Once a bid is accepted by a CareTaker, all bids by the same PetOwner, with the same Pet on the same date for other caretakers will be rejected automatically. This rejection is then applied to the entire transaction.
    e. If a caretaker becomes fully booked after approving a bid, all other bids pending for this caretaker on that date should be rejected as well.
    f. If a bid has been rejected, it cannot be approved later.

21) Rating and Review
    a. The Rating and Review can be filled by the PetOwner only for approved bids.
    b. Once a rating is set, it cannot be changed anymore.
    c. There can only be one review and rating for each complete transaction (defined as a range of consecutive days of service for a particular pet) with a CareTaker.

    d. If the update of a rating causes the average rating of a CareTaker to fall such that their upper bound on their fee per day for each pet type falls, check if their existing fee per day exceeds the upper bound and reset it to base price if so.
    e. If the update of a rating causes the max pet limit of a part-time CareTaker to fall (due to lower average rating), then some previously available dates (for booking a service) are no longer available, reject all pending bids for the unavailable dates.

22) When leave is successfully applied, or CareTaker account is deleted, pending or rejected bids on the affected Availability dates should have its data saved in an Invalidated Bids table as a single transaction before being deleted from Bidding Records.
23) When a Pet or PetOwner account is deleted, the affected bids should be transferred to a BidsWithoutPetOwner table. Because rating, review and bid price are still important to be tracked for the CareTaker it should be regarded as part of the Bidding Records.
24) A particular Pet, PetOwner account or CareTaker account cannot be deleted if there are still approved bids in the future (after the current date) yet to be completed. (This guarantees each entry to BidsWithoutPetOwner has a unique primary key because the date will be different even if a PetOwner with the same username is deleted and created multiple times)
25) The PCS Admin account cannot be deleted if there are caretakers being managed by that admin account.

Annex 1.2

Constraints Not Enforced by Relational Schema

2) There are two types of Users,
    a. They are either Consumers or PCS Admin(Covering Constraint)
    b. And cannot be both (Non-Overlapping Constraint).
3) There are two types of Consumers,
    a. They are either PetOwner or CareTaker (Covering Constraint).
10) CareTakers have to be able to take care of at least one Pet Type.
11) Each CareTaker will also have a Fee per Day specified (by them) for the services of caring for each Pet Type.
    a. This fee is greater than or equals the base price of the category of Pet Type.
    b. Fee per Day can be priced up to 1.1 times of base price if the CareTaker has an averaged rating of 4 or higher, and up to 1.2 times of base price if the CareTaker has an average rating of 4.7 or higher as the upper limit, otherwise the upper limit is equal to the base price.
12) When the PCS Admin updates the base price for a particular pet category, check if the existing fee per day for any caretaker for that category exceeds the upper limit, or falls below the base price. If so, reset to new base price.
15) Part-Time CareTakers have to update their available dates for the next two years.
    a. They also cannot apply for leave.
16) FullTimers can apply for Leave Days to be taken. These Leave days allow their Availability dates (working days) to be removed from consideration, their Available dates for the year will be tracked (their leave dates are simply dates not in Available dates). There are two conditions that have to be satisfied for the new Leave to be approved.
    a. They have no Pet booked to care for during that date.
    b. Their leave should not violate the minimum 2x150 consecutive days of work for the year.
19) PetOwners may bid for CareTaker's services on behalf of their Pet on the days the particular CareTaker is available.
    a. PetOwners can only bid for available dates in the future (after current date)
    b. A PetOwner may bid for a range of consecutive dates for the Available dates of the CareTaker.
        i. This will create individual bid entries for each of the consecutive Available dates bidded for, with the same values for all the other attributes for the bidding record.
        ii. These multiple bids for consecutive dates should all have the same End Date as an indication that the bid for these multiple bids across a range of dates should be considered together as a single transaction.
    c. A CareTaker's services are only open for bidding for a new bid, if they are Available(working) on a particular day and have at least 1 remaining pet slot.
    d. The PetOwner should also bid only for CareTakers that can care for their Pet's Type for their bid to be valid.
    e. If the Pet of a particular PetOwner already has an approved bid for a given date, he/she cannot bid for anymore services for that same pet on that same date.
    g. Every bid must have a bid price greater than or equal to the CareTaker's fee per day for that pet category.
    h. Each bid will appear in the Bidding Records with a pending status initially.
    i. Each bid will initially have null values for review and rating.
    j. Users who are both caretaker and PetOwners may not bid for their own services.
20) CareTakers will review the bids and accept the bids according to his preference.
    a. Updating a bid to approved status by the CareTaker should only be for bids on dates after the current date.
    b. A bid can only be approved by the CareTaker if he/she is available to take at least one more pet without exceeding his/her max pet limit.
    c. A CareTaker can only reject or accept all the bids from a Pet and PetOwner within a single transaction as a group, that is, the set of bids with a common end date.
    d. Once a bid is accepted by a CareTaker, all bids by the same PetOwner, with the same Pet on the same date for other caretakers will be rejected automatically. This rejection is then applied to the entire transaction.
    e. If a caretaker becomes fully booked after approving a bid, all other bids pending for this caretaker on that date should be rejected as well.
    f. If a bid has been rejected, it cannot be approved later.
21) The Bidding Records could have a Rating (integer from 1 to 5) and Review (string of text) by the PetOwner regarding the service provided by the CareTaker.
    a. The Rating and Review can be filled by the PetOwner only for approved bids.
    b. Once a rating is set, it cannot be changed anymore.
    c. There can only be one review and rating for each complete transaction (defined as a range of consecutive days of service for a particular pet) with a CareTaker.
    d. If the update of a rating causes the average rating of a CareTaker to fall such that their upper bound on their fee per day for each pet type falls, check if their existing fee per day exceeds the upper bound and reset it to base price if so.

   e. If the update of a rating causes the max pet limit of a part-time CareTaker to fall (due to lower average rating), then some previously available dates (for booking a service) are no longer available, reject all pending bids for the unavailable dates.

22) When leave is successfully applied, or CareTaker account is deleted, pending or rejected bids on the affected Availability dates should have its data saved in an Invalidated Bids table as a single transaction before being deleted from Bidding Records.
23) When a Pet or PetOwner account is deleted, the affected bids should be transferred to a BidsWithoutPetOwner table. Because rating, review and bid price are still important to be tracked for the CareTaker it should be regarded as part of the Bidding Records.
24) A particular Pet, PetOwner account or CareTaker account cannot be deleted if there are still approved bids in the future (after the current date) yet to be completed. (This guarantees each entry to BidsWithoutPetOwner has a unique primary key because the date will be different even if a PetOwner with the same username is deleted and created multiple times)

```sql
----------------------------------------------------------------------
--------------- Annex 2.0 Helper Functions For Triggers ---------------
----------------------------------------------------------------------
/*To track today's date, currently uses dummy*/
CREATE OR REPLACE FUNCTION currentDate()
RETURNS DATE AS $$
    BEGIN
        RETURN (SELECT CAST('2019-12-31'AS DATE));
        /* dummy current date. For live application,
        it should returns (SELECT CURRENT_DATE)*/
    END; $$
LANGUAGE plpgsql;

/*Returns a table combining the two tables Bids and BidsWithoutPetOwner*/
/*While not strictly necessary, this function reduces
    complexity of code to make it more readible*/
    /*To be used only in triggers*/
CREATE OR REPLACE FUNCTION combinedBids()
RETURNS TABLE(
    petowner VARCHAR,
    petname VARCHAR,
    caretaker VARCHAR,
    avail DATE,

    edate DATE,
    transferType VARCHAR,
    paymentType VARCHAR,
    price FLOAT(4),
    isPaid BOOLEAN,
    status CHAR(1),
    review VARCHAR,
    rating INTEGER
    ) AS $$
    BEGIN
        RETURN QUERY (SELECT * FROM Bids UNION
                                SELECT * FROM BidsWithoutPetOwner);
    END;$$
LANGUAGE plpgsql;

/*Maps the average rating of a caretaker to the
    max price multiplier which when multiplied by
    base price is the upper bound of fee per day*/
CREATE OR REPLACE FUNCTION
    mapAvgRatingToMultiplier(avgratings NUMERIC)
RETURNS NUMERIC AS $$
    BEGIN
        IF avgratings>4.7 THEN
            RETURN 1.2;
        END IF;
        IF avgratings>4.0 THEN
            RETURN 1.1;
        END IF;
        RETURN 1.0;
    END; $$
LANGUAGE plpgsql;

/*Takes in the caretaker's username as input and computes
    the corresponding max price multiplier which when
    multiplied by base price is the upper bound of fee per day
    for a caretaker*/
CREATE OR REPLACE FUNCTION computeMaxPriceMultiplier(ctname VARCHAR)
RETURNS NUMERIC AS $$
    DECLARE avgratings NUMERIC;
    BEGIN
        IF ctname NOT IN (SELECT caretaker FROM combinedBids() ) THEN
            RETURN 1.0;
        END IF;
        SELECT AVG(B.rating) INTO avgratings FROM combinedBids() B
            WHERE B.caretaker=ctname;
        RETURN (SELECT mapAvgRatingToMultiplier(avgratings));
    END; $$
LANGUAGE plpgsql;

```

```
/*Takes in the caretaker's username and a new rating as input and
    computes the new max price multiplier after the rating is
    added into the database*/
CREATE OR REPLACE FUNCTION
    computeUpdatedMaxPriceMultiplier(ctname VARCHAR, newrating INTEGER)
RETURNS NUMERIC AS $$
    DECLARE sumratings NUMERIC;
    DECLARE numratings NUMERIC;
    DECLARE avgratings NUMERIC;
    BEGIN
        IF ctname NOT IN (SELECT caretaker FROM combinedBids() ) THEN
            RETURN (SELECT mapAvgRatingToMultiplier(newrating));
        END IF;
        SELECT SUM(B.rating) INTO sumratings
            FROM combinedBids() B WHERE B.caretaker=ctname;
        SELECT COUNT(B.rating) INTO numratings
            FROM combinedBids() B WHERE B.caretaker=ctname;
        sumratings := sumratings + newrating;
        numratings := numratings + 1;
        avgratings := sumratings/numratings;
        RETURN (SELECT mapAvgRatingToMultiplier(avgratings));
    END; $$
LANGUAGE plpgsql;

/*Takes in the caretaker's username as input and computes the
    max number of pet he or she can care for in a day*/
CREATE OR REPLACE FUNCTION computeMaxPet(ctname VARCHAR)
RETURNS INTEGER AS $$
    DECLARE avgratings NUMERIC;
    BEGIN
        IF ctname NOT IN (SELECT C.username FROM CareTakers C) THEN
            RAISE EXCEPTION 'no such caretaker in database';
            RETURN -1;
        END IF;
        IF ctname NOT IN (SELECT P.username FROM PartTimers P) THEN
            RETURN 5;
        END IF;
        SELECT AVG(B.rating) INTO avgratings
            FROM combinedBids() B WHERE B.caretaker=ctname;
        IF avgratings>4.7 THEN
            RETURN 5;
        END IF;
        IF avgratings>4.0 THEN
            RETURN 4;
        END IF;
        IF avgratings>3.0 THEN
            RETURN 3;
        END IF;
        RETURN 2.0;
    END; $$
LANGUAGE plpgsql;

/*Takes in the caretaker's username as input and computes the
    new max number of pet he or she can care for in a day
    after the new rating is added into the database*/
CREATE OR REPLACE FUNCTION
    computeUpdatedMaxPet(ctname VARCHAR, newrating INTEGER)
RETURNS INTEGER AS $$
    DECLARE sumratings NUMERIC;
    DECLARE numratings NUMERIC;
    DECLARE avgratings NUMERIC;
    BEGIN
        IF ctname NOT IN (SELECT C.username FROM CareTakers C) THEN
            RAISE EXCEPTION 'no such caretaker in database';
            RETURN -1;
        END IF;
        IF ctname NOT IN (SELECT P.username FROM PartTimers P) THEN
            RETURN 5;
        END IF;
        SELECT SUM(B.rating) INTO sumratings
            FROM combinedBids() B WHERE B.caretaker=ctname;
        SELECT COUNT(B.rating) INTO numratings
            FROM combinedBids() B WHERE B.caretaker=ctname;
        sumratings := sumratings + newrating;
        numratings := numratings + 1;
        avgratings := sumratings/numratings;
        IF avgratings>4.7 THEN
            RETURN 5;
        END IF;
        IF avgratings>4.0 THEN
            RETURN 4;
        END IF;
        IF avgratings>3.0 THEN
            RETURN 3;
        END IF;
        RETURN 2.0;
    END; $$
LANGUAGE plpgsql;

/*Takes in CareTaker Username, the datebooked, and the max pet
    for the CareTaker as inputs and checks if
    the careTaker is available to care for at least
    one more pet on the datebooked*/
CREATE OR REPLACE FUNCTION
    isAvailable(ctname VARCHAR, datebooked DATE, maxpet INTEGER DEFAULT NULL)
RETURNS BOOLEAN AS $$
    BEGIN
        IF maxpet IS NULL THEN
            maxpet := (SELECT computeMaxPet(ctname));
        END IF;
        RETURN (SELECT COUNT(*)
                    FROM combinedBids() B
                    WHERE B.caretaker=ctname
                        AND B.avail=datebooked AND B.status='a'
                    )<maxpet
                AND
                datebooked IN (SELECT A.avail
                                FROM Availability A
                                WHERE A.caretaker=ctname);
    END; $$
LANGUAGE plpgsql;

/*Takes in the petcategory as input and returns the base price
    associated with that category*/
CREATE OR REPLACE FUNCTION getBasePrice(petcategory VARCHAR)
RETURNS NUMERIC AS $$
    BEGIN
        RETURN (SELECT P.baseprice FROM PetTypes P WHERE P.category=petcategory);
    END; $$
LANGUAGE plpgsql;
```

```
----------------------------------------------------------------------
---------------- Annex 2.1 Check Is Valid Bid Trigger ---------------
----------------------------------------------------------------------
DROP TRIGGER IF EXISTS checkIsValidBid ON Bids;
CREATE OR REPLACE FUNCTION isValidBid()
RETURNS TRIGGER AS $$
    DECLARE minfee NUMERIC;
    DECLARE petcategory VARCHAR;
    DECLARE oldmult NUMERIC;
    DECLARE newmult NUMERIC;
    DECLARE isupdate BOOLEAN;
    DECLARE oldmaxpet INTEGER;
    DECLARE newmaxpet INTEGER;
    BEGIN
        isupdate := (OLD.petowner IS NOT NULL AND OLD.petname IS NOT NULL
            AND OLD.caretaker IS NOT NULL AND OLD.avail IS NOT NULL);
        IF NOT isupdate THEN /*Insertion of a new bid*/
            IF NEW.avail<=(SELECT currentDate()) THEN
                RAISE EXCEPTION 'bids should only be for dates tomorrow
                    onwards (i.e. greater than current date)';
                RETURN NULL;
            END IF;
            IF NEW.caretaker=NEW.petowner THEN
                RAISE EXCEPTION 'a user who is both a caretaker and
                    petowner cannot bid for his own services';
                RETURN NULL;
            END IF;
            IF NOT (SELECT isAvailable(NEW.caretaker, NEW.avail)) THEN
                RAISE EXCEPTION 'caretaker is unavailable on that date';
                RETURN NULL;
            END IF;
            SELECT P.category INTO petcategory
                FROM Pets P WHERE P.petowner=NEW.petowner
                    AND P.petname=NEW.petname;
            IF petcategory NOT IN (
                    SELECT A.category
                    FROM AbleToCare A
                    WHERE A.caretaker=NEW.caretaker) THEN
                RAISE EXCEPTION 'caretaker is unable to care for this pet type';
                RETURN NULL;
            END IF;
            SELECT A.feeperday INTO minfee
                FROM AbleToCare A WHERE A.caretaker=NEW.caretaker
                    AND A.category=petcategory;
            IF NEW.price<minfee THEN
                RAISE EXCEPTION 'bid price is below caretakers
                        fee per day for that pet';
                RETURN NULL;
            END IF;
            IF NEW.status!='p' OR NEW.rating IS NOT NULL
                    OR NEW.review IS NOT NULL THEN
                RAISE EXCEPTION 'initial status and rating for bid
                        should be pending and null';
                RETURN NULL;
            END IF;
            IF EXISTS (
                    SELECT *
                    FROM combinedBids() B
                    WHERE B.petowner=NEW.petowner AND B.petname=NEW.petname
                        AND B.avail=NEW.avail AND B.status='a') THEN
                RAISE EXCEPTION 'pet already has an existing appointment
                        with another caretaker on this date';
                RETURN NULL;
            END IF;
            RETURN NEW;
        END IF;
        /*Is an update*/
        IF NEW.status!=OLD.status THEN /*status update*/
        /*When updating to approve bid*/
```

```
            IF NEW.status='a' THEN
                IF NEW.avail<=(SELECT currentDate()) THEN
                    RAISE EXCEPTION 'approved bids should only be for dates
                            tomorrow onwards (i.e. greater than current date)';
                    RETURN NULL;
                END IF;
                /*When updating to approve a bid, check if number of pets
                        taken for that day is within max pet limit*/
                IF NOT (SELECT isAvailable(NEW.caretaker, NEW.avail)) THEN
                    RAISE EXCEPTION 'this date is fully booked, cannot approve';
                    RETURN NULL;
                END IF;
                IF OLD.status='r' THEN /*check the bid is not already rejected*/
                    RAISE EXCEPTION 'cannot approve already rejected bid';
                    RETURN NULL;
                END IF;
                /*When updating to approve bid, all other bids pending for
                        the pet on this same date will be rejected*/
                /*Note when we reject a bid on an avail date, all other
                        similar bids with same edate should also be rejected*/
                UPDATE Bids SET status='r'
                    WHERE petowner=NEW.petowner AND petname=NEW.petname
                        AND caretaker!=NEW.caretaker AND status='p'
                        AND NEW.edate IN
                            (SELECT B.edate
                                FROM Bids B
                                WHERE B.petowner=NEW.petowner
                                    AND B.petname=NEW.petname AND B.status='p'
                                    AND B.avail=NEW.avail);
                /*If caretaker becomes fully booked, all other bids pending
                        for this caretaker on this date will be rejected*/
                /*Note when we reject a bid on an avail date, all other
                        similar bids with same edate should also be rejected*/
                IF (SELECT COUNT(*)
                        FROM combinedBids() B
                        WHERE B.caretaker=NEW.caretaker
                            AND B.avail=NEW.avail AND B.status='a'
                        )=(SELECT computeMaxPet(NEW.caretaker)-1) THEN
                    UPDATE Bids SET status='r'
                        WHERE caretaker=NEW.caretaker AND edate!=NEW.edate
                            AND status='p'
                            AND edate IN
                                (SELECT B.edate
                                    FROM Bids B
                                    WHERE B.caretaker=NEW.caretaker
                                        AND B.avail=NEW.avail AND
                                        B.status='p');
                END IF;
            END IF;
        END IF;
        /*Update on rating*/
        IF NEW.rating!=OLD.rating
                OR (NEW.rating IS NOT NULL AND OLD.rating IS NULL)
                OR (NEW.rating IS NULL AND OLD.rating IS NOT NULL) THEN
            IF OLD.avail!=OLD.edate THEN
                RAISE EXCEPTION 'Can only rated once for each transaction,
                        the availability should equal end date for
                        the bid to be rated';
                RETURN NULL;
            END IF;
            IF NEW.rating IS NOT NULL THEN
                IF OLD.status!='a' THEN
                    RAISE EXCEPTION 'cannot update ratings on bid that is
                            not approved';
                    RETURN NULL;
                END IF;
                IF OLD.rating IS NOT NULL THEN
                    RAISE EXCEPTION 'cannot update rating that has been set already';
                    RETURN NULL;
                END IF;
                /*Update fees per day if average ratings change*/
                SELECT computeMaxPriceMultiplier(OLD.caretaker) INTO oldmult;
                SELECT computeUpdatedMaxPriceMultiplier(OLD.caretaker, NEW.rating)
                        INTO newmult;
                IF newmult!=oldmult THEN
                    UPDATE AbleToCare SET feeperday=(SELECT getBasePrice(category))
                        WHERE caretaker=OLD.caretaker
                            AND feeperday>newmult*(SELECT getBasePrice(category));
                END IF;
                IF OLD.caretaker IN (SELECT P.username FROM PartTimers P) THEN
                    /*Only part-timers*/
                    SELECT computeMaxPet(OLD.caretaker) INTO oldmaxpet;
                    SELECT computeUpdatedMaxPet(OLD.caretaker, NEW.rating)
                            INTO newmaxpet;
                    /*if max pets decrease, some previously available dates no longer
                    available
                     *  then have to update pending bids on unavailable dates*/
                    IF newmaxpet<oldmaxpet THEN
                        UPDATE Bids SET status='r'
                            WHERE caretaker=OLD.caretaker AND status='p'
                                AND edate IN
                                (SELECT B.edate
                                    FROM Bids B
                                    WHERE B.caretaker=caretaker AND B.status='p'
                                        AND NOT (SELECT isAvailable(caretaker, avail,
                                        newmaxpet)) );
                    END IF;
                END IF;
            ELSE /*new rating is null and old rating is not null*/
                RAISE EXCEPTION 'cannot update rating that has been set already to null';
                RETURN NULL;
            END IF;
        END IF;
        /*update on review*/
        IF NEW.review!=OLD.review OR (NEW.review IS NOT NULL AND OLD.review IS NULL)
                OR (NEW.review IS NULL AND OLD.review IS NOT NULL) THEN
            IF OLD.avail!=OLD.edate THEN
                RAISE EXCEPTION 'Can only review once for each transaction,
                        the availability should equal end date for the bid to be
                        reviewed';
                RETURN NULL;
            END IF;
        END IF;
        RETURN NEW;
    END; $$
LANGUAGE plpgsql;

CREATE TRIGGER checkIsValidBid
BEFORE INSERT OR UPDATE ON Bids
FOR EACH ROW EXECUTE PROCEDURE isValidBid();
```

```
-----------------------------------------------------------------
---------------- Annex 2.2 Check Is Valid Leave Trigger ---------------
-----------------------------------------------------------------
DROP TRIGGER IF EXISTS checkIsValidLeave ON Availability;
CREATE OR REPLACE FUNCTION isValidLeave()
RETURNS TRIGGER AS $$
    DECLARE yearofleave INTEGER;
    DECLARE isvalid150days BOOLEAN;
    DECLARE error VARCHAR;
    BEGIN
        IF OLD.caretaker NOT IN (SELECT username FROM caretakers) THEN /*This means
        delete cascade from caretakers is called*/
            RETURN OLD; /*bypass all the validLeave conditions*/
        END IF; /*This condition to ensure on delete cascade from caretakers still work*/
        IF OLD.caretaker IN (SELECT P.username FROM PartTimers P) THEN
            RAISE EXCEPTION 'part-timers cannot take leave';
            RETURN NULL;
        END IF;
        IF OLD.avail IN (SELECT B.avail FROM combinedBids() B WHERE
        B.caretaker=OLD.caretaker AND B.status='a') THEN
            RAISE EXCEPTION 'caretaker is occupied that day';
            RETURN NULL;
        END IF;
        yearofleave := (SELECT EXTRACT(YEAR FROM OLD.avail));
        isvalid150days := (SELECT COUNT(*)>=2 OR (COUNT(*)>=1 AND
        MAX(D.enddate-D.startdate+1)>=300)
                    FROM
                        (SELECT X.avail AS startdate, X.avail+MIN(Y.avail-X.avail) AS
                        enddate
                        FROM
                            (SELECT AV.avail
                            FROM Availability AV
                            WHERE AV.caretaker=OLD.caretaker AND
                            AV.avail!=CAST(OLD.avail AS DATE)
                                AND (SELECT EXTRACT(YEAR FROM AV.avail))=yearofleave
                                AND AV.avail-1 NOT IN (SELECT A.avail
                                                        FROM Availability A
                                                        WHERE A.caretaker=OLD.caretaker
                                                        AND A.avail!=CAST(OLD.avail AS
                                                        DATE)
                                                        AND (SELECT EXTRACT(YEAR FROM
                                                        A.avail))=yearofleave) ) X
                            INNER JOIN
                            (SELECT AV.avail
                            FROM Availability AV
                            WHERE AV.caretaker=OLD.caretaker AND
                            AV.avail!=CAST(OLD.avail AS DATE)
                                AND (SELECT EXTRACT(YEAR FROM AV.avail))=yearofleave
                                AND AV.avail+1 NOT IN (SELECT A.avail
                                                        FROM Availability A
                                                        WHERE A.caretaker=OLD.caretaker
                                                        AND A.avail!=CAST(OLD.avail AS
                                                        DATE)
                                                        AND (SELECT EXTRACT(YEAR FROM
                                                        A.avail))=yearofleave) ) Y
                        ON X.avail<= Y.avail
                        GROUP BY X.avail ) D
                    WHERE D.enddate-D.startdate+1>=150 );
        IF NOT isvalid150days THEN
            RAISE EXCEPTION 'violates 150 days constraint';
            RETURN NULL;
        END IF;

        /*place pending or rejected bids affected by leave into invalidated bids*/

        INSERT INTO InvalidatedBids(petowner, petname, caretaker, sdate, edate,
        transferType, paymentType, price)
        SELECT X.petowner, X.petname, X.caretaker, X.avail, X.edate, Y.transferType,
        Y.paymentType, Y.price
        FROM
            (SELECT D.petowner, D.petname, D.caretaker, MIN(D.avail) AS avail, D.edate
            FROM Bids D
            WHERE caretaker=OLD.caretaker AND status!='a'
                AND edate IN
                (SELECT B.edate
                FROM Bids B
                WHERE B.caretaker=caretaker AND B.status!='a' AND B.avail=OLD.avail )
            GROUP BY D.petowner, D.petname, D.caretaker, D.edate) X
            NATURAL JOIN Bids Y
        WHERE NOT EXISTS (SELECT *
                          FROM InvalidatedBids I
                          WHERE I.petowner=X.petowner AND I.petname=X.petname
                          AND I.caretaker=X.caretaker AND I.edate = X.edate)
        ;

        /*delete bids on leave dates*/

        DELETE FROM Bids
            WHERE caretaker=OLD.caretaker AND status='p'
                AND edate IN
                (SELECT B.edate
                FROM Bids B
                WHERE B.caretaker=caretaker AND B.status='p' AND B.avail=OLD.avail );
        RETURN OLD;
    END; $$
LANGUAGE plpgsql;

CREATE TRIGGER checkIsValidLeave
BEFORE DELETE ON Availability
FOR EACH ROW EXECUTE PROCEDURE isValidLeave();
```

```
------------------------------------------------------------------------
---------------- Annex 2.3 Check Is Deletable Caretaker ----------------
------------------------------------------------------------------------
DROP TRIGGER IF EXISTS checkIsDeletableCareTaker ON CareTakers;
CREATE OR REPLACE FUNCTION isDeletableCareTaker()
RETURNS TRIGGER AS $$
    BEGIN
        IF EXISTS (SELECT * FROM combinedBids() B
                WHERE B.caretaker=OLD.username AND B.status='a' AND (SELECT
                currentDate())<=B.avail) THEN
            RAISE EXCEPTION 'this caretaker is servicing a petowner in the future!';
            RETURN NULL;
        END IF;

        /*place pending or rejected bids affected by removal of caretaker into
        invalidated bids*/
        INSERT INTO InvalidatedBids(petowner, petname, caretaker, sdate, edate,
        transferType, paymentType, price)
        SELECT X.petowner, X.petname, X.caretaker, X.avail, X.edate, Y.transferType,
        Y.paymentType, Y.price
        FROM
            (SELECT D.petowner, D.petname, D.caretaker, MIN(D.avail) AS avail, D.edate
            FROM Bids D
            WHERE caretaker=OLD.username AND status!='a'
            GROUP BY D.petowner, D.petname, D.caretaker, D.edate) X
            NATURAL JOIN Bids Y
        WHERE NOT EXISTS (SELECT *
                          FROM InvalidatedBids I
                          WHERE I.petowner=X.petowner AND I.petname=X.petname
                          AND I.caretaker=X.caretaker AND I.edate = X.edate)
        ;

        RETURN OLD;
    END; $$
LANGUAGE plpgsql;

CREATE TRIGGER checkIsDeletableCareTaker
BEFORE DELETE ON CareTakers
FOR EACH ROW EXECUTE PROCEDURE isDeletableCareTaker();
```

```sql
-----------------------------------------------------------------------
---------------- Annex 2.4 Finding Availability of CareTakers ---------------
-----------------------------------------------------------------------
/* Support the browsing for caretakers by pet owners */
/* Note that the inputs are variable depending on what the pet owner selects from the
front end*/
/* input: start date, end date, pet category */
/* output: caretaker, pet category, atc.feeperday, caretaker average rating, number of
ratings, start date, end date */
CREATE OR REPLACE FUNCTION
        viewAvailability(startdatein DATE, enddatein DATE, petcategory VARCHAR)
RETURNS TABLE(
    caretaker VARCHAR,
    category VARCHAR,
    feeperday FLOAT(4),
    avgrating NUMERIC,
    numratings BIGINT,
    startdate DATE,
    enddate DATE
    ) AS $$
    BEGIN
        RETURN QUERY (
            SELECT atc.caretaker, atc.category, atc.feeperday,
                AVGRC.avgrating, AVGRC.numratings,
                startdatein AS startdate, enddatein AS enddate
            FROM AbleToCare atc
                NATURAL JOIN ( /*Table of each caretaker's average
                                rating and number of rating*/
                    SELECT CB.caretaker, AVG(CB.rating) AS avgrating,
                        COUNT(CB.rating) AS numratings
                    FROM (SELECT * FROM Bids UNION
                            SELECT * FROM BidsWithoutPetOwner) CB
                    GROUP BY CB.caretaker
                    UNION
                    SELECT CT.username AS caretaker,
                        NULL AS avgrating, 0 AS numratings
                    FROM CareTakers CT
                    WHERE CT.username NOT IN (
                        SELECT CB1.caretaker
                        FROM (SELECT * FROM Bids UNION
                                SELECT * FROM BidsWithoutPetOwner) CB1
                        )
                    ) AVGRC
            WHERE EXISTS
                (SELECT 1 FROM
                    (SELECT availCT.caretaker, COUNT(*) AS days
                        FROM /*table of caretaker's availability
                                to take on another pet on that date, within a
                                restricted date range*/
                            (SELECT AV.caretaker, AV.avail
                                FROM /*table of tabulating number of pets a caretaker
                                    has to take care on a work day*/
                                    (SELECT B.caretaker, B.avail, COUNT(*) AS cnt
                                        FROM Bids B
                                        WHERE B.status ='a'
                                        GROUP BY B.caretaker, B.avail
                                        UNION
                                        SELECT A.caretaker, A.avail, 0 AS cnt
                                        FROM Availability A
                                        WHERE NOT EXISTS (
                                            SELECT *
                                            FROM Bids B
                                            WHERE B.caretaker=A.caretaker
                                                AND B.avail=A.avail AND B.status='a'
                                            )
                                    ) AV
                                WHERE AV.cnt<(/*This nested query computes
                                            the max pet limit of AV.caretaker*/
                                                SELECT CASE
```

```
                                                   WHEN AV.caretaker NOT IN (
                                                       SELECT P.username FROM
                                                       PartTimers P
                                                       ) THEN 5
                                                       /*means caretaker is full-time,
                                                       default value 5.
                                                       Look at part-time case below*/
                                                   WHEN AVGR.avgrating>4.7 THEN 5
                                                   WHEN AVGR.avgrating>4.0 THEN 4
                                                   WHEN AVGR.avgrating>3.0 THEN 3
                                                   ELSE 2
                                                   END
                                               FROM (SELECT AVG(CB.rating) AS avgrating
                                                       FROM (SELECT * FROM Bids UNION
                                                           SELECT * FROM
                                                           BidsWithoutPetOwner) CB
                                                       WHERE
                                                       CB.caretaker=AV.caretaker) AVGR
                                               )
                                   AND (AV.avail BETWEEN startdatein AND enddatein)
                                   /*restrict to date range*/
                           )AS availCT
                   GROUP BY availCT.caretaker
                   HAVING COUNT(*) = (enddatein-startdatein) +1
                   /*means caretaker is available evey day within the date range*/
                   ) AS t
               WHERE t.caretaker = ATC.caretaker
               )
               AND atc.category = petcategory
           ORDER BY avgrating DESC NULLS LAST, numratings DESC, feeperday ASC
       );
   END;$$
LANGUAGE plpgsql;

/*Example query using the function*/
SELECT * FROM viewAvailability((SELECT CAST('2020-01-01' AS DATE)), (SELECT
CAST('2020-01-06' AS DATE)), 'dog');
```

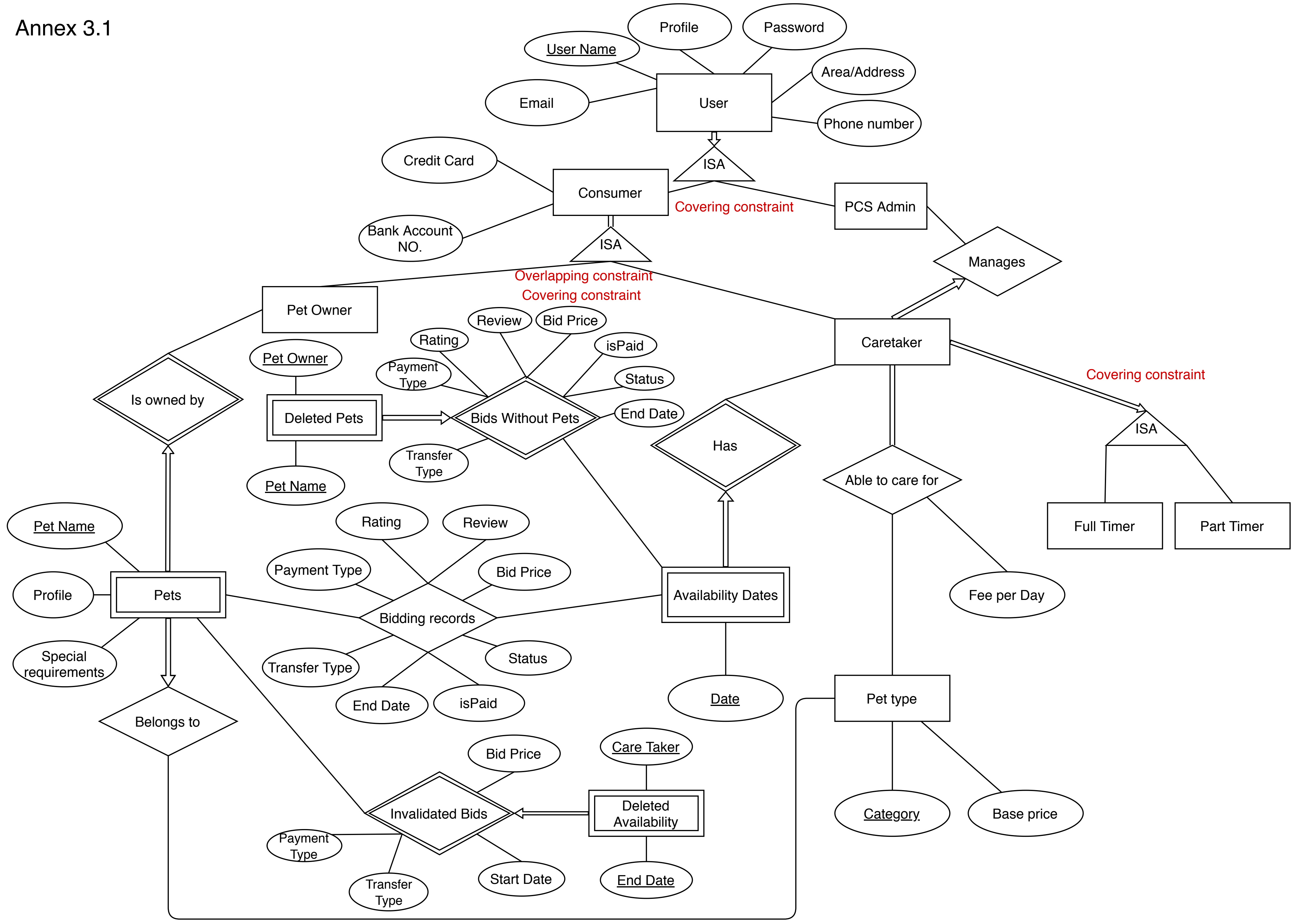
```
-----------------------------------------------------------------------------
--------------- Annex 2.5 Browsing Caretaker's Wage and Pet Days ---------------
-----------------------------------------------------------------------------
/* Support browsing of Caretaker's wages and pet days clocked for a
    particular month with caretaker's average ratings and num ratings*/
/* input:datein (should correspond to the month of interest)*/
/* output: caretaker, contract type, salary, pet days clocked,
    average rating, number of ratings */
CREATE OR REPLACE FUNCTION viewCareTakersWagePetDaysRatings(datein DATE)
RETURNS TABLE(
    caretaker VARCHAR,
    contract TEXT,
    salary FLOAT,
    petdaysclocked BIGINT,
    avgrating NUMERIC,
    numratings BIGINT
    ) AS $$
    BEGIN
        RETURN QUERY (
            SELECT WG.caretaker, WG.contract, WG.salary,
                    WG.petdaysclocked, RT.avgrating, RT.numratings
            FROM
                (/*This table tabulates the salary and pet days clocked
                    for part-timers with at least 1 approved paid bid*/
                SELECT CB.caretaker, 'Part-Time' AS contract,
                        SUM(price)*0.75 AS salary, COUNT(*) AS petdaysclocked
                FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
                WHERE CB.avail <= CAST(date_trunc('month', datein)
                        + interval '1 month - 1 day' AS DATE)
                    AND CB.avail >= CAST(date_trunc('month', datein) AS DATE)
                    AND CB.isPaid = TRUE
                    AND CB.status = 'a'
                    AND CB.caretaker IN (SELECT PT.username FROM PartTimers PT)
                GROUP BY CB.caretaker
                UNION
                /*This table tabulates the salary and pet days
                    clocked for part-timers with no approved paid bid*/
                SELECT PT.username AS caretaker, 'Part-Time' AS contract,
                        0.0 AS salary , 0 AS petdaysclocked
                FROM PartTimers PT
                WHERE PT.username NOT IN (
                        SELECT B.caretaker
                        FROM (SELECT * FROM Bids UNION SELECT * FROM
                        BidsWithoutPetOwner) B
                        WHERE B.avail <= CAST(date_trunc('month', datein)
                                + interval '1 month - 1 day' AS DATE)
                            AND B.avail >= CAST(date_trunc('month', datein) AS DATE)
                            AND B.isPaid = TRUE
                            AND B.status = 'a')
                UNION
                /*This table tabulates the salary and pet days clocked for
                    full-timers with at least 1 approved paid bid*/
                SELECT CT.username AS caretaker, 'Full-Time' AS contract, (
                        /*computation of salary for one full-timer*/
                        SELECT CASE
                                WHEN SUM(OFS.price) IS NOT NULL
                                    THEN 3000+SUM(OFS.price)*0.8
                                ELSE 3000
                                END
                        FROM ( /*Find prices of all excess pet days*/
                            SELECT *
                            FROM (SELECT * FROM Bids UNION
                                    SELECT * FROM BidsWithoutPetOwner) CB
                            WHERE CB.caretaker=CT.username
                                AND CB.avail <= CAST(date_trunc('month', datein)
                                        + interval '1 month - 1 day' AS DATE)
                                AND CB.avail >= CAST(date_trunc('month', datein) AS DATE)
                                AND CB.isPaid = TRUE
                                AND CB.status = 'a'
                            ORDER BY CB.price ASC
                            OFFSET 60  /*First 60 entries are assigned to
                                        the base pet days*/
                            ) OFS
                    ) AS salary,
                    ( /*counting the pet days clocked for this full-timer*/
                    SELECT COUNT(*) AS petdaysclocked
                    FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
                    WHERE CB.caretaker=CT.username
                        AND CB.avail <= CAST(date_trunc('month', datein)
                                + interval '1 month - 1 day' AS DATE)
                        AND CB.avail >= CAST(date_trunc('month', datein) AS DATE)
                        AND CB.isPaid = TRUE
                        AND CB.status = 'a'
                    ) AS petdaysclocked
                FROM CareTakers CT
                WHERE CT.username NOT IN (SELECT PT.username FROM PartTimers PT)
                    /*condition to select only full-timers*/
                ) WG
                NATURAL JOIN (
                /*This table computes the average ratings for each caretaker*/
                SELECT CB.caretaker, AVG(CB.rating) AS avgrating,
                        COUNT(CB.rating) AS numratings
                FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
                GROUP BY CB.caretaker
                UNION
                SELECT CT.username AS caretaker, NULL AS avgrating, 0 AS numratings
                FROM CareTakers CT
                WHERE CT.username NOT IN (
                        SELECT CB1.caretaker
                        FROM (SELECT * FROM Bids UNION SELECT * FROM
                        BidsWithoutPetOwner) CB1)
                ) RT
            ORDER BY RT.avgrating ASC, RT.numratings ASC, WG.petdaysclocked ASC
            /*order by average ratings, number of ratings, then pet days clocked*/
        );
    END;$$
LANGUAGE plpgsql;

/*Example query using the function*/
SELECT * FROM viewCareTakersWagePetDaysRatings(CAST('2020-02-01' AS DATE));
```

```sql
-----------------------------------------------------------------------------
---------------- Annex 2.6 Browsing Monthly Business Financials ---------------
-----------------------------------------------------------------------------
/* Support browsing of key financial information about the business
    by PCS adminsuch as profit, revenue, salary cost and pet serviced
    for each months over a range of months*/
/*Input: startdate, enddate
    (corresponding to the start month and end month of interest)*/
/*Output: year, month, total profit, revenue,
    total salary paid and total pets for each month, uses CTE*/
CREATE OR REPLACE FUNCTION viewMonthlyFinancials(startdatein DATE, enddatein DATE)
RETURNS TABLE(
    year_ FLOAT,
    month_ FLOAT,
    profit FLOAT,
    totalrevenue REAL,
    totalsalarypaid FLOAT,
    totalpets BIGINT
    ) AS $$
    BEGIN
        RETURN QUERY (
            WITH transactions AS (
                SELECT CB.petowner, CB.petname, CB.caretaker,
                    CB.avail, CB.price,
                    EXTRACT(MONTH FROM CB.avail) AS month_,
                    EXTRACT(YEAR FROM CB.avail) AS year_
                FROM (SELECT * FROM Bids UNION SELECT * FROM BidsWithoutPetOwner) CB
                WHERE CB.status='a' AND CB.isPaid
                    AND CB.avail <=CAST(date_trunc('month', enddatein)
                            + interval '1 month - 1 day' AS DATE)
                    AND CB.avail >=CAST(date_trunc('month', startdatein) AS DATE)
                )
            SELECT TSP.year_, TSP.month_,
                TRP.totalrevenue-TSP.totalsalarypaid AS profit,
                TRP.totalrevenue, TSP.totalsalarypaid, TRP.totalpets
            FROM (/*This table tabulates the total salary paid for each month*/
                SELECT CTS.year_, CTS.month_,
                    SUM(CTS.salary) AS totalsalarypaid
                FROM ( /*This table tabulates the salary for each caretaker
                        for a given month and year*/
                    SELECT CT.username AS caretaker, MY.year_, MY.month_,
                        (SELECT CASE /*compute salary for each caretaker*/
                            /*Full-timers case*/
                            WHEN CT.username NOT IN
                                (SELECT PT.username FROM PartTimers PT)
                            THEN
                            (SELECT CASE
                                WHEN SUM(OFS.price) IS NOT NULL
                                    THEN 3000+SUM(OFS.price)*0.8
                                ELSE 3000
                                END
                            FROM (
                                SELECT *
                                FROM transactions CB
                                WHERE CB.caretaker=CT.username AND CB.year_=MY.year_
                                    AND CB.month_=MY.month_
                                ORDER BY CB.price ASC
                                OFFSET 60
                                ) OFS
                            )
                        ELSE /*part-timers case*/
                            (
                            SELECT CASE
                                WHEN SUM(price) IS NOT NULL THEN SUM(price)*0.75
                                ELSE 0
                                END
                            FROM transactions CB
                            WHERE CB.caretaker=CT.username AND CB.year_=MY.year_
                                AND CB.month_=MY.month_
                                )
                            END
                            )AS salary
                        FROM CareTakers CT,
                            (SELECT DISTINCT T0.month_,
                                T0.year_ FROM transactions T0) MY
                        ) AS CTS
                    GROUP BY CTS.year_, CTS.month_
                    ) TSP
                    NATURAL JOIN
                    ( /*This table tabulates the total revenue and the
                        total number pets served for each month*/
                    SELECT TR0.year_, TR0.month_, SUM(TR0.price) AS totalrevenue,
                            COUNT(DISTINCT CONCAT(TR0.petowner, ',', TR0.petname) ) AS
                            totalpets
                    FROM transactions TR0
                    GROUP BY TR0.year_, TR0.month_
                    ) TRP
                ORDER BY TSP.year_ ASC, TSP.month_ ASC
            );
        END;$$
    LANGUAGE plpgsql;

    /*Example query using the function*/
    SELECT * FROM viewMonthlyFinancials(CAST('2019-01-01' AS DATE), CAST('2020-07-05' AS
    DATE));
```

Annex 3.1


User Name
Profile
Password
Email
User
Area/Address
Phone number
ISA
Credit Card
Consumer
Covering constraint
PCS Admin
Bank Account NO.
ISA
Manages
Overlapping constraint
Covering constraint
Pet Owner
Caretaker
Review
Bid Price
Rating
isPaid
Pet Owner
Payment Type
Status
Is owned by
Deleted Pets
Bids Without Pets
End Date
Covering constraint
Has
ISA
Transfer Type
Pet Name
Able to care for
Full Timer
Part Timer
Pet Name
Rating
Review
Payment Type
Bid Price
Profile
Pets
Availability Dates
Fee per Day
Bidding records
Special requirements
Transfer Type
Status
End Date
isPaid
Date
Pet type
Belongs to
Bid Price
Care Taker
Invalidated Bids
Deleted Availability
Category
Base price
Payment Type
Transfer Type
Start Date
End Date