

AY20/21 Semester 1

CS2102 Database Systems

Project Report

Group 10

Keng Jun Xian (A0167718R)

Labi Trisha Angelica Vergara (A0185430J)

Ng Shi Han (A0171605L)

Phuah Wei Ke (A0171391H)

Phoon Jia Juin (A0185683N)

**Table of Contents**

1. **Project Responsibilities**
   1. Preliminary constraints and ER diagram
   2. Application
   3. Report
2. **Description of Application**
   1. Application's Data Requirements
   2. Application’s Data Functionalities
   3. Application’s Data Constraints
   4. Interesting/non trivial Application’s Functionalities/Implementation
3. **ER Model of Application**
   1. ER Model
   2. Application’s constraints not captured by ER Model
4. **Relational Schema**
   1. Relational Schema
   2. Application’s constraints not enforced by Relational Schema
5. **Form of Database (BCNF or 3NF)**
   1. Functional Dependencies
   2. Check for BCNF
   3. Check for 3NF
6. **Three interesting triggers in the Application**
7. **SQL code of three of the most complex queries**
8. **Specification of the software tools/frameworks used**
   1. ER Diagram
   2. Application
9. **Representative screenshots of application in action.**
10. **Summary of difficulties encountered and lessons learned from the project**
11. **Project Responsibilities**
    1. Preliminary constraints and ER diagram

As a group, we discussed the constraints and came up with the relational schema.

After working on the relational schema, we designed the ER diagram together via an online flowchart maker.

* 1. Application

We discussed and decided on what we wanted to be included in our application. Jun Xian being the one with the knowledge on how to implement the application, taught everyone how to use nodeJS and VSCode. Together we built the application.

For the triggers, everyone tried to come up with at least one. We then proceeded to choose the top three interesting and feasible ones.

We decided on the queries that we felt are useful. Similarly to the report, we created a google document to write them.

* 1. Report

A google document was created and used to write our report. Everyone did their part by writing up the various sections.

1. **Description of Application**
   1. Application's Data Requirements

The test data can be loaded in PostgreSQL with pgAdmin and the PCSDB.tar file provided. To connect to the database in express, navigate to server/db.js and replace the password with your personal database password.

* 1. Application’s Data Functionalities

This application enables a pet owner to book any caretaker by using the bidding system. The caretaker can use the same application to accept bookings. This dual nature is accomplished by keeping track of a user's role in the application's state. The user can be either PetOwner or Caretaker, or both.

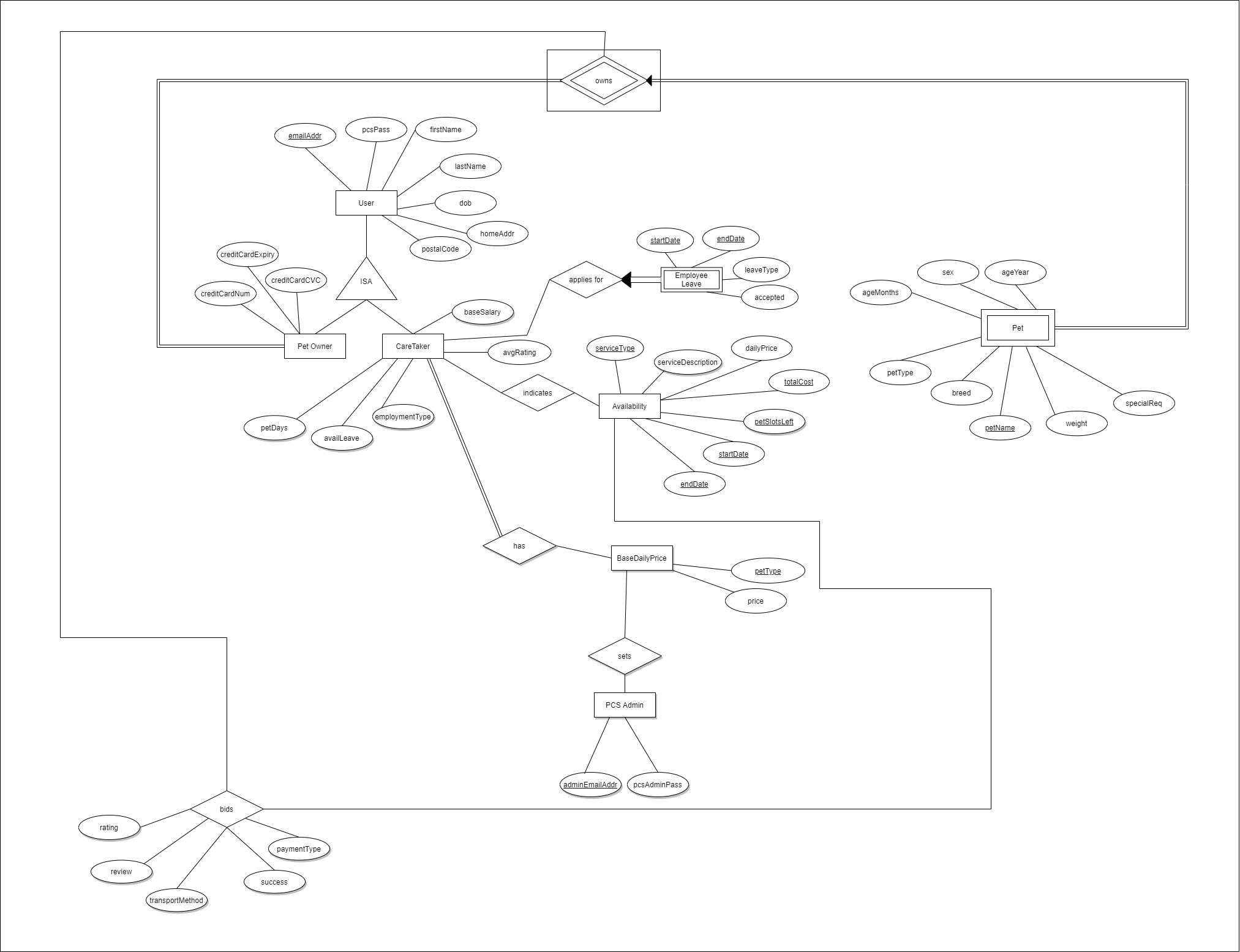
* 1. Application’s Data Constraints

Stripe is typically used to handle payments and check if a given credit card is valid. However, due to it being complex to integrate and also difficult to produce valid credit card numbers for our test data, our app does not check for validity of a given credit card.

* 1. Interesting/non trivial Application’s Functionalities/Implementation

Our PCSAdmin portal is hosted on the same site as the app. It is accessible via a specific link, and once logged in, the global state will handle different views based on the type of user that is logged in, who is either a normal user or an admin. In retrospect, perhaps we could have merged PCSAdmin into PCSUser (adding Admin to the ISA relationship), and used a 'role' attribute to differentiate between User and Admin accounts.

1. **ER Model of Application**
   1. ER Model



* 1. Application’s constraints not captured by ER Model

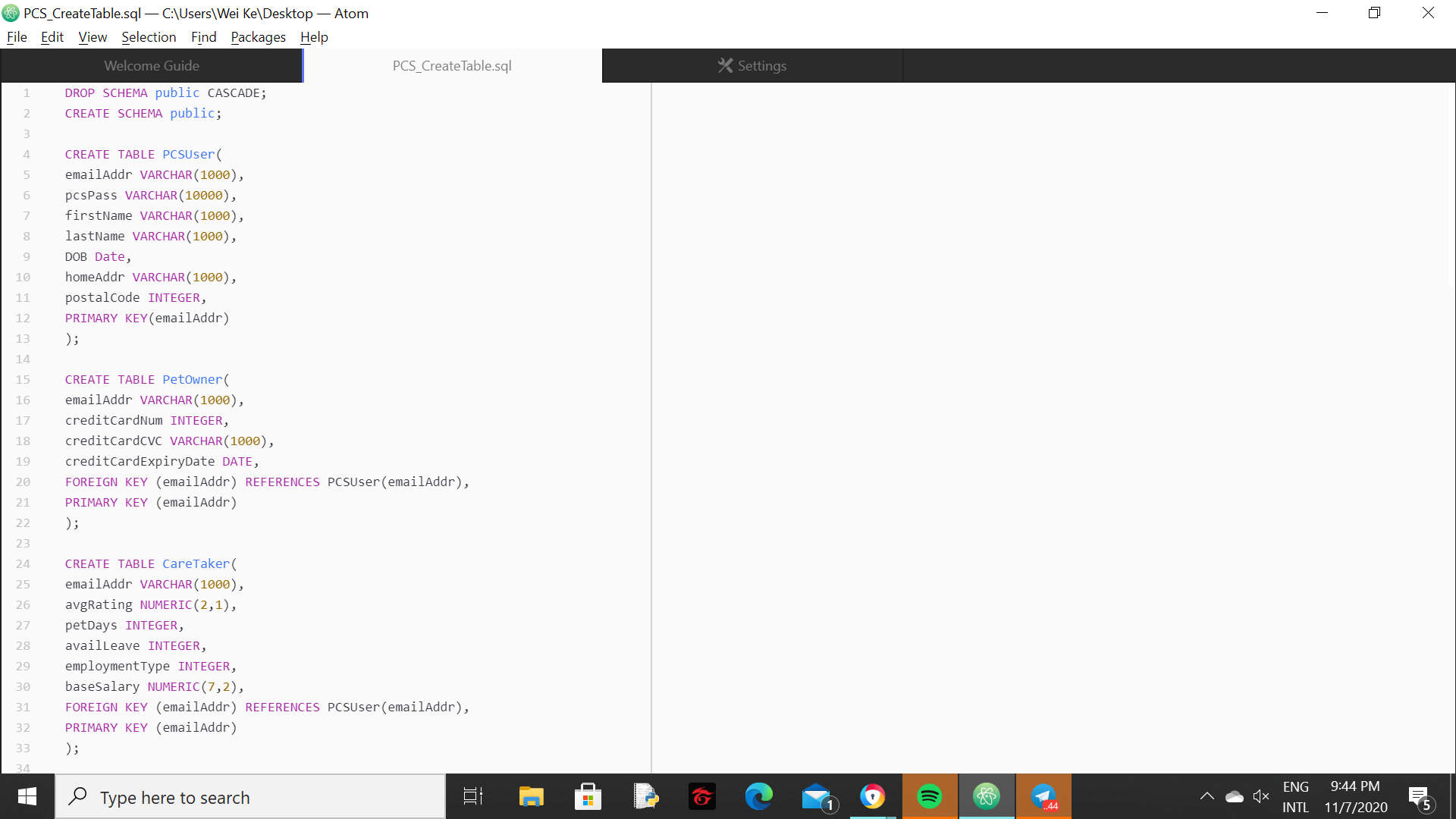
1. Constraint maximum value of ‘petSlotsLeft’ based on avgRating of each CareTaker.

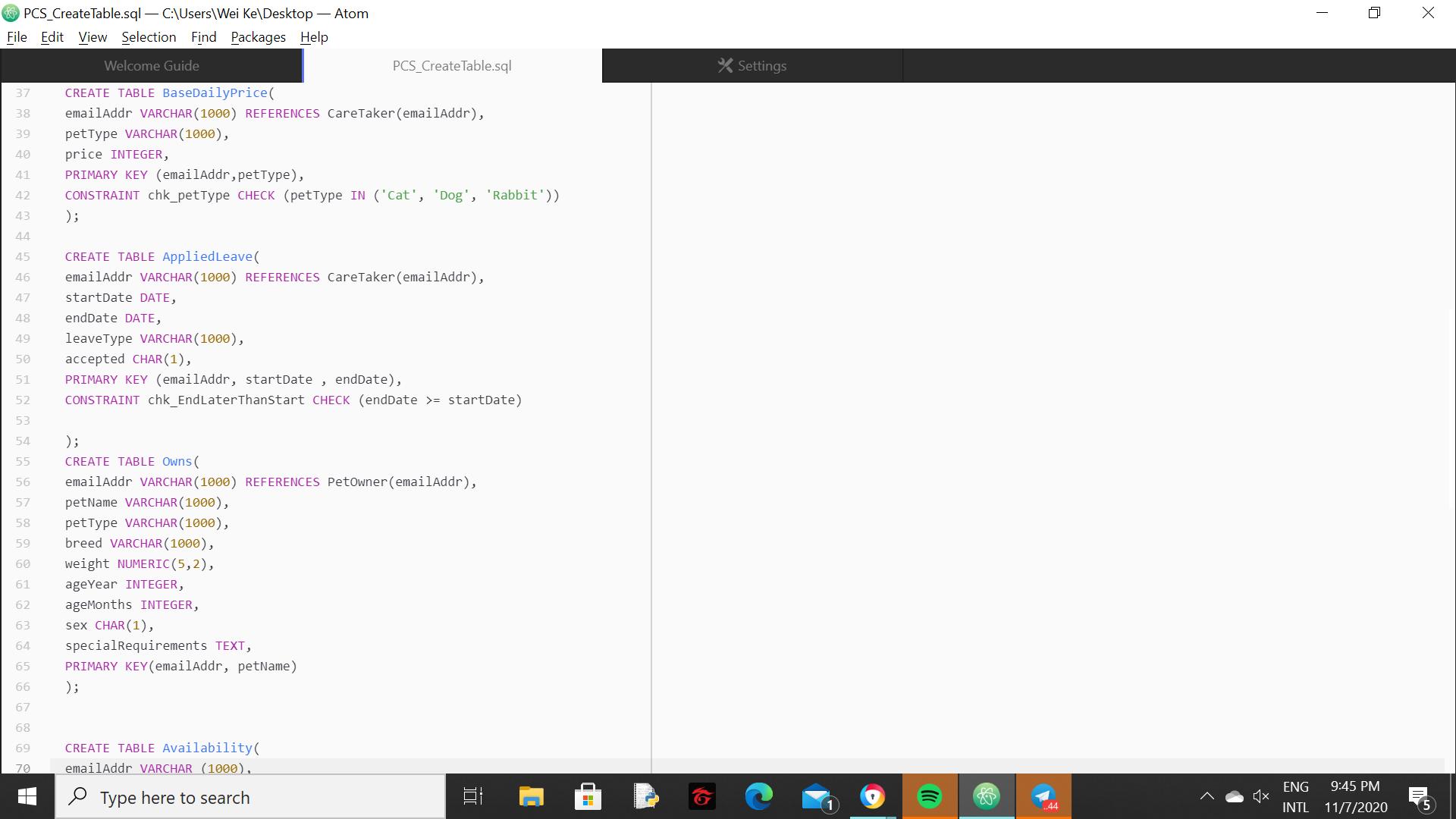
2. PetOwner can only bid for the service that their pet corresponds to, i.e. can only bid for DogSittingService only if PetOwner owns a dog.

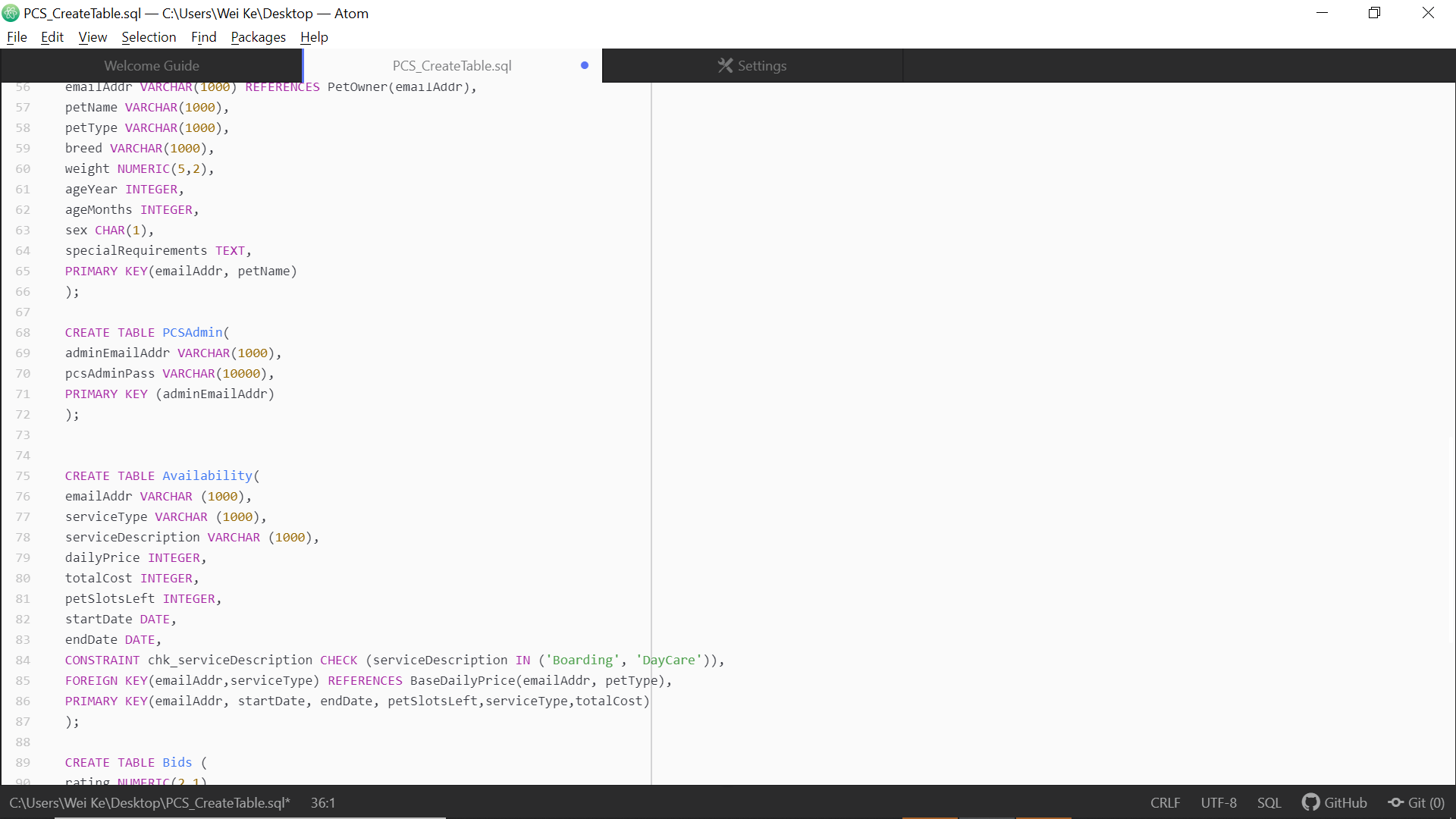
3. For a pet that is involved in an accepted service, the pet owner cannot accept another service for the same pet in the same time period.

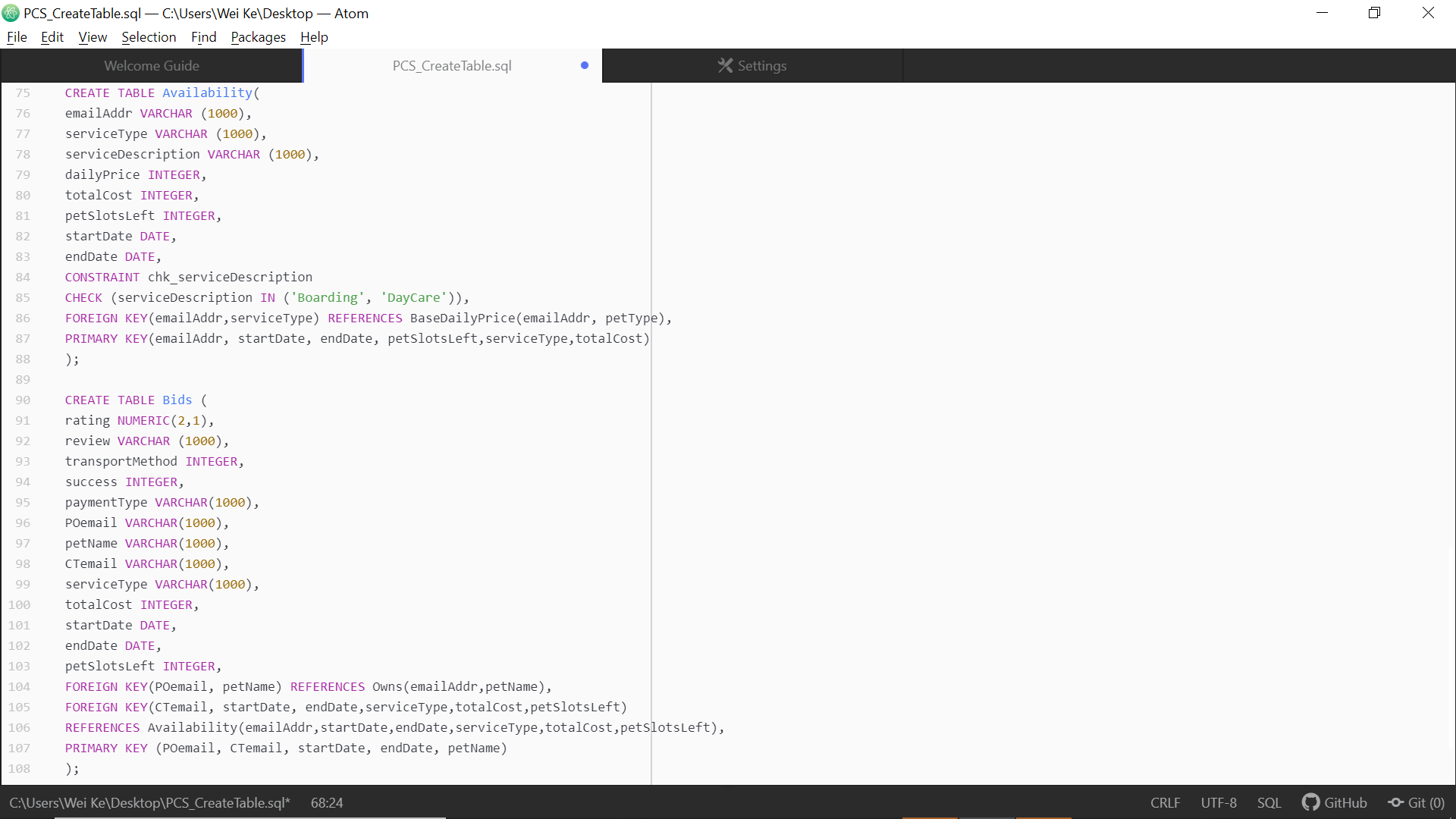
4. All ISA hierarchy: covering and no overlap.

1. **Relational Schema**
   1. Relational Schema







****

* 1. Application’s constraints not enforced by Relational Schema

1. In the app, the user cannot be both a normal user and a PCSAdmin. However, this is not enforced by the relational schema. In the app, we check that the email address of a new user does not match any records in either the PCSUser table or the PCSAdmin table. However, this constraint is not enforced in the relational schema.

2. When a CareTaker in on Applied Leave, the listing of a new Caretaker Availability should be prevented.

3. Checking the validity of Applied Leave, and removal of Availability to accommodate leave period.

4. After a successful bid, the available slots for the caretaker should be updated automatically.

Points 2, 3, and 4 are not enforced by our relational schema and will be done so by triggers in section 6.

1. **Form of Database**
   1. Functional Dependencies

First, we list out all functional dependencies available from our relationship schema R.

adminEmailAddr -> {pcsAdminPass}

emailAddr -> {pcsPass, firstName,lastName, DOB, homeAddr,creditCard,creditCardType, avgRating, petDays, availLeave, employmentType, baseSalary}

emailAddr, petType -> {price}

emailAddr, petName -> {petType, breed, weight, ageYear, ageMonths, sex, specialRequirements}

emailAddr, startDate, endDate -> {leaveType, accepted}

emailAddr, startDate, endDate, petSlotsLeft, serviceType, totalCost -> {serviceDescription, dailyPrice}

emailAddr (POemail), emailAddr (CTemail), startDate, endDate, petName -> {rating, review, transportMethod, success, paymentType, serviceType, totalCost, petSlotsLeft}

* 1. Check for BCNF

For a relationship R with a set of Functional Dependencies F to be in a BCNF, every Functional Dependency of the form a -> A in F+ must fulfil either of the following conditions:

1. a -> A is trivial
2. a is a superkey

Check adminEmailAddr -> {pcsAdminPass}

* Not a trivial Functional Dependency
* adminEmailAddr + = adminEmailAddr, pcsAdminPass

As adminEmailAddr is not a superkey, R is not in BCNF

* 1. Check for 3NF

Now, we check to see if our relationship R is in 3NF. For a relationship R with a set of Functional Dependencies F to be in a 3NF, every Functional Dependency of the form a -> A in F+ must fulfil either of the following conditions:

1. a -> A is trivial
2. a is a superkey
3. A is a prime attribute (i.e. A is part of any key)

Identifying attributes within possible keys in R:

|  |  |  |
| --- | --- | --- |
| LHS | Both | RHS |
| adminEmailAddr  emailAddr  petName  startDate  endDate | petType  petSlotsLeft  serviceType  totalCost | pcsAdminPass  pcsPass  firstName  lastName  DOB  homeAddr  creditCard  creditCardType  avgRating  petDays  availLeave  employmentType  baseSalary  Price  Breed  Weight  ageYear  ageMonths  Sex  specialRequirements  leaveType  Accepted  serviceDescription  dailyPrice  Rating  Review  transportMethod  Success  paymentType |

Those under ‘LHS’ and ‘Both’ will be part of possible keys, while those under ‘RHS’ will never be part of a key.

Check adminEmailAddr -> {pcsAdminPass}

* Not a trivial Functional Dependency
* adminEmailAddr + = adminEmailAddr, pcsAdminPass
* pcsAdminPass is not part of any key

As the relation does not fulfil any condition, R is also not in 3NF.

Individually, not all of our tables are in BCNF. In other words, for a dependency A → B, there are cases where A is a non-prime attribute while B is a prime attribute. To resolve this, it is usually common to further decompose our tables. However, this is avoided as this will result in more complex queries between multiple tables. We already have many complex queries and having such tables may only make querying more time consuming. Furthermore, if the relation has no proper decomposition, then it may lead to problems like loss of information and our group would like to avoid that as far as possible.

An example of such a table is the PetOwner table. As emailAddr is the primary key, it is a prime attribute. However, all credit card numbers are unique globally, and hence a possible functional dependency {creditCardNum -> emailAddr} exists. However, creditCardNum is not a prime attribute. We can then see that this table is not in BCNF.

1. **Three interesting triggers in the Application**

**Trigger 1: Preventing the listing of a new Caretaker Availability when the Caretaker is on Applied Leave**

This trigger is inserted into the Availability table. This table represents the Caretaker’s listed availability which Pet Owners can bid for. This trigger is implemented to enforce the constraint that a Caretaker should not be listed as available during his applied leave, and subsequently a Petowner should not be able to bid for a job where the Caretaker is on applied leave. This is done by checking the start and end dates of the availability against the leave dates for the specific caretaker. An exception is raised for invalid Availability listings.

The implementation is as follows:

****

**Trigger 2: Checking the validity of Applied Leave, and removal of Availability to accommodate leave period.**

This trigger is inserted into the AppliedLeave table. This table represents the Caretaker’s applied leave periods. This trigger is implemented to enforce two constraints.

First, a Caretaker that has already accepted a bid for a job request should not have leave in the same period approved as there would be a contradiction in dates. This is done by checking for the presence of any accepted bid within the leave Start and End dates for the specific caretaker. An exception is raised for invalid approved Leave Applications.

Secondly, a Caretaker that has his leave approved should not have his availability period overlapping with his leave period. This is done by checking the start and end dates of the availability against the leave dates for the specific caretaker if the approved leave does not violate the first constraint. The relevant availability period that lies within the leave period is removed to prevent PetOwners from bidding for it. An exception is also raised to notify Caretakers of the deletion.

The implementation is as follows:





**Trigger 3: Automatic updating of Caretaker petSlotsLeft after a successful bid**

This trigger is inserted into the Bids table. This table represents the bids Petowners has made to obtain the services of the Caretakers. Caretaker’s applied leave periods. This trigger is implemented to check for the validity of a Successful Bid, and subsequently update the relevant tables on a Caretaker’s petSlotsLeft in the Availability and Bids table. This is done by checking the success and petSlotsLeft attribute in the Availability table. An exception will be raised if we attempt a successful bid when petSlotsLeft is 0, else the petSlotsLeft will be reduced by one.

The implementation is as follows:





1. **SQL code of three of the most complex queries**

**Query 1: Total number of Pet taken care of in the month**

SELECT COUNT(\*)

FROM bids

WHERE EXTRACT(MONTH FROM startdate) = “Month Value in INTEGER”;

e.g. March = 3

Query 1 supports the browsing summary information for PCS Administrator. It allows access to the total number of pets being taken care of in a given month.

**Query 2: The month with the highest number of newly accepted jobs**

WITH months AS

SELECT (DATE\_PART(‘month’,startDate)) AS month FROM Bids WHERE success = 1

SELECT month,COUNT(\*)

FROM months GROUP BY month ORDER BY COUNT(\*) DESC LIMIT 1;

Query 2 supports the browsing summary information for PCS Administrator. It allows access to the month with the highest number of newly accepted bids.

**Query 3: Average Age of Petowner per Pet Type**

WITH POAge AS(

SELECT EXTRACT(YEAR FROM NOW()) - EXTRACT(YEAR FROM DOB) AS AGE, p.emailAddr FROM PCSUser c, PetOwner p WHERE c.emailAddr= p.emailAddr)

SELECT AVG(age)

FROM Owns O LEFT JOIN POAge P ON O.emailAddr = P.emailAddr

GROUP BY petType;

Query 3 supports the browsing summary information for Pet Owner. It gives the average age of Pet Owners by the respective Pet Types.

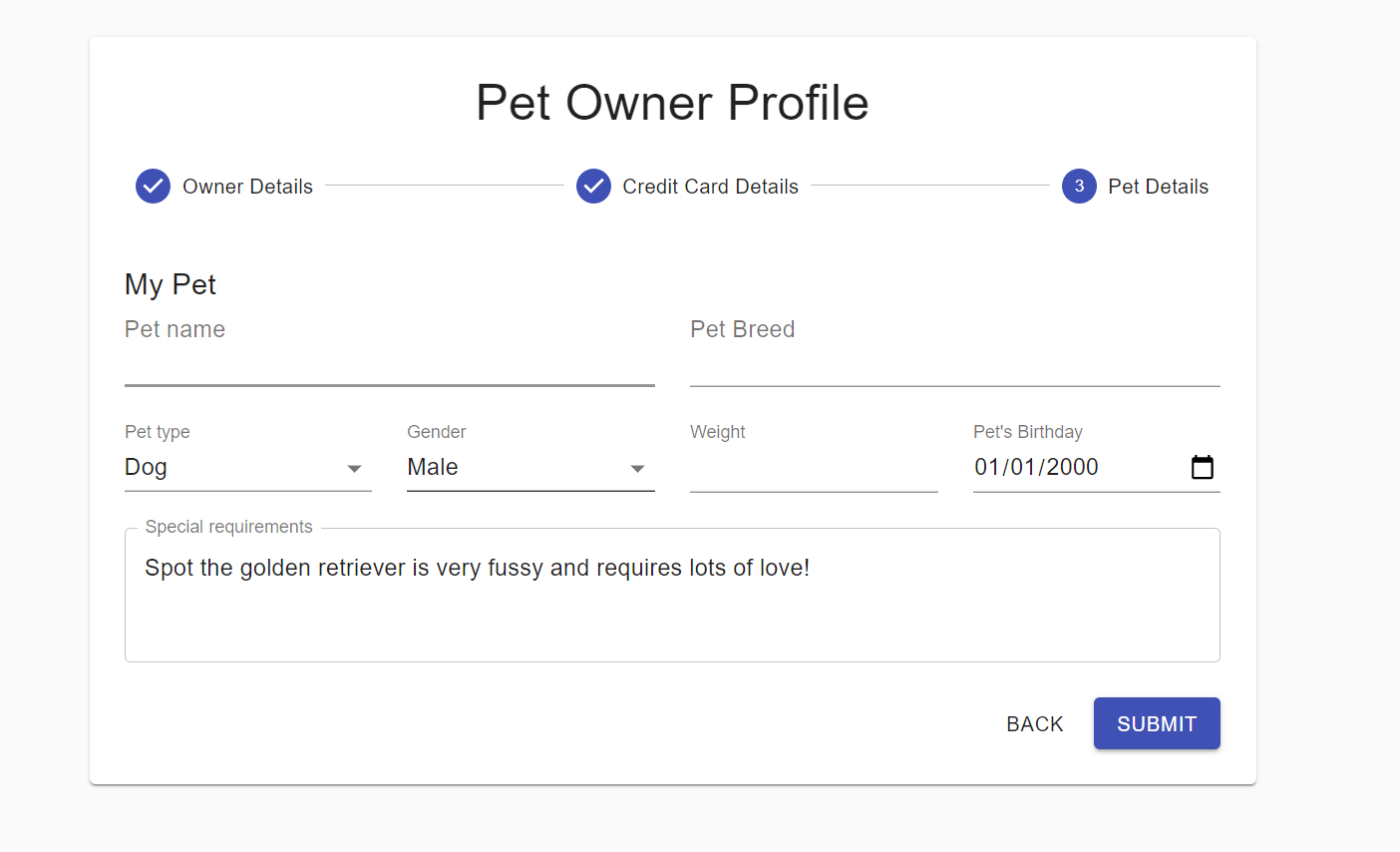
1. **Specification of the software tools/frameworks used**
   1. ER Diagram

Diagrams.net, a free online diagram application and flowchart maker was used to build our ER diagram.

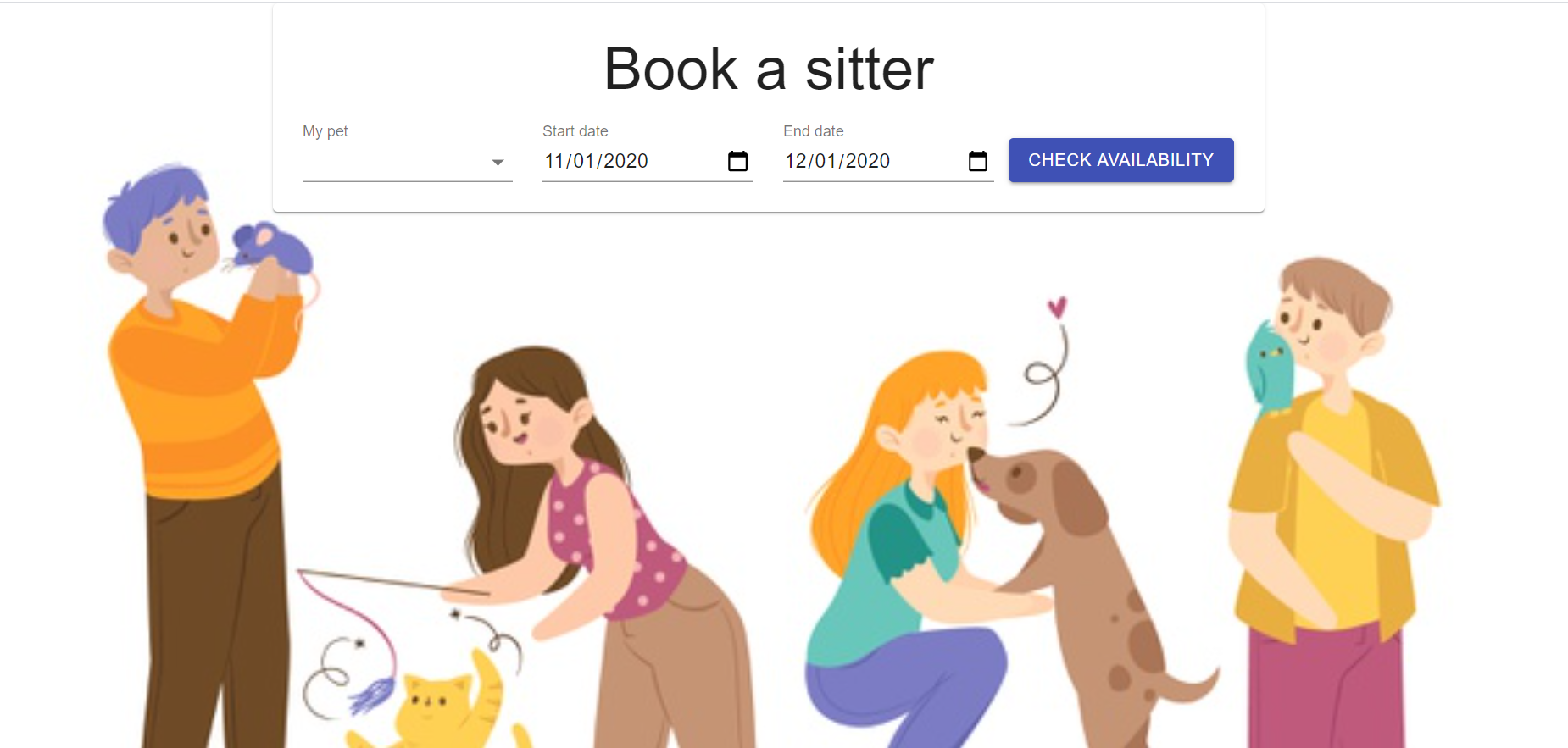
* 1. Application

The software tools used are PostgreSQL, NodeJS express library, React, and VisualStudio Code.

1. **Representative screenshots of application in action.**

**Representative screenshot 1: Creation of Pet Owner’s Profile**

**Representative screenshot 2: Submitting bookings for a Caretaker**

****

1. **Summary of difficulties encountered and lessons learned from the project**

We faced a lot of difficulties during the implementation of the application. For instance, this project is the first time where most of us are introduced to NodeJS and Visual Studio Code. For the very few who have been introduced to the above, the knowledge was limited. As a result, we had to spend quite a bit of time learning the basic knowledge of these applications. Also, we faced some technical difficulties such as fatal error while installing postgre, having difficulties to link the connection between front end and back end application.

Furthermore, most of us are not fluent in coding due to all our majors not falling into the Faculty of Computing. This was definitely a factor that made us struggle to keep up with the short deadline of implementing our application.

From the project, we learnt the basics of implementation from the database tier to the application tier and after which the client tier whereby the website can be used. Also, we learnt the importance of having a good ER diagram as it forms the foundation for the building of the application.