Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

I have read and I understand the plagiarism provisions in the General Regulations of the University

Student Name	Tianze Zhang
Student ID Number	19334401
Course Title	Computer Science
Module Title	CSU34041: Information Management II
Lecturer(s)	Vincent Wade
Title	Information Management II Assessment: Pet Clinic Report
Date Submitted	29/03/2022
Word Count	1500

Contents

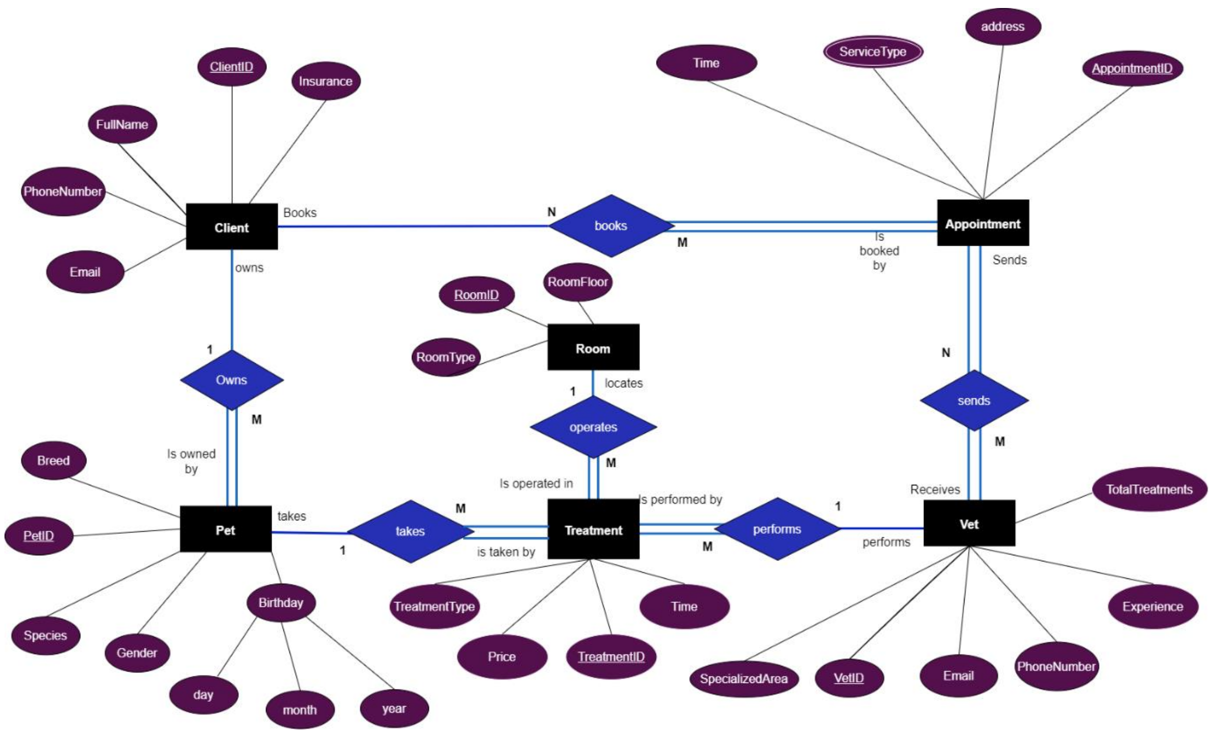
1 Description of database Application area and ER Model

1.1 Application Description

Jonathon Wall is an experienced veterinarian and opens a pet clinic. This pet clinic allows the customers to book appointments with the vets directly, and the appointments are organised by the veterinarians. The business is expanding quickly and as the increasing of the clients and the appointments, it gets harder for the vets to organise all of the information. Therefore, Johnathon decides to create a database to keep track of all the information of the pet clinic.

There are multiple clients booking multiple appointments with the clinic on a regular basis. Every client owning 0 or more pets can book an appointment, which mean clients can book appointments with pets owned by other people. However, the pet must have an owner. All the appointments are sent by the client and then reviewed by different vets working in the clinic. The vets then perform treatments according to the appointment. Each vet can perform multiple treatments. Each pet can take multiple treatments. All treatments must be operated in a room, and each room can be used for different treatments.

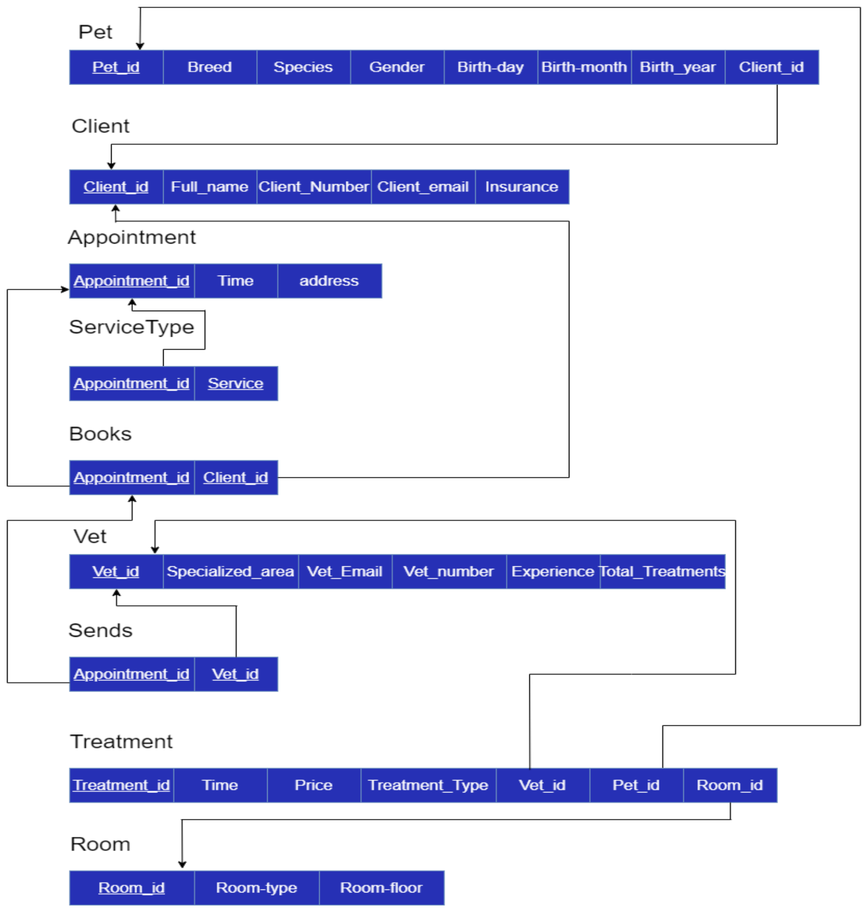
1.2 Entity Relationship Diagram



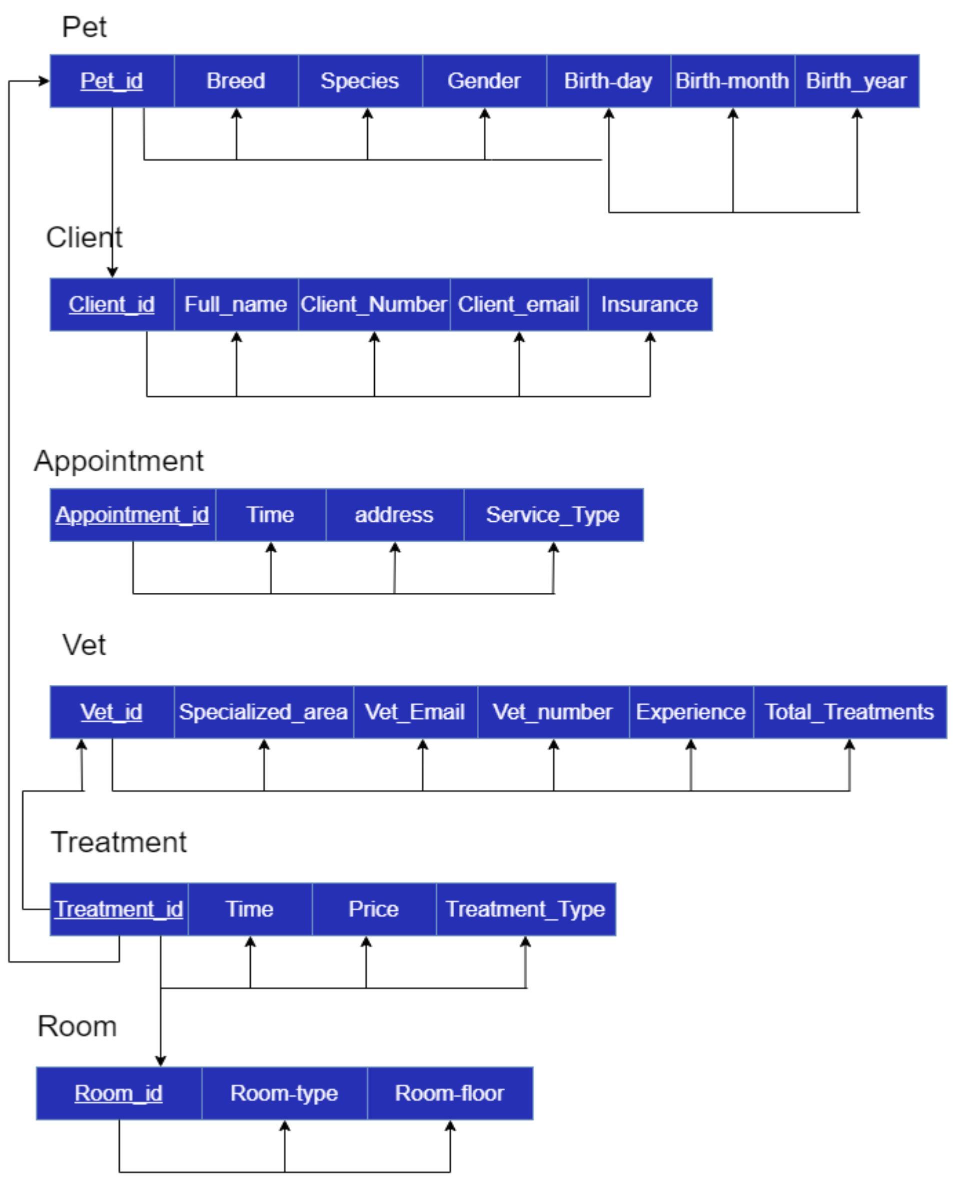
1.2.1 Assumptions Made

- Each pet has a unique ID. Similarly, there are IDs for Client, Appointment, Vet, Treatment and Room.
- Appointments must be booked by a client.
- A client can decide if they will make the appointment for the pet or not, (client doesn't have to make an appointment).
- All appointments must be sent to the vets.
- All appointments must be received by all the vets
- All treatments must be taken by a pet and performed by a vet.
- The vet might not perform the treatment (if something special happened and he/she cannot perform at the appointed time etc).
- The pet also might not be able to take the treatment

1.3 Mapping to Relational Schema



1.4 Functional Dependency Diagrams (for proposed relations)



2 Explanation of data and SQL code

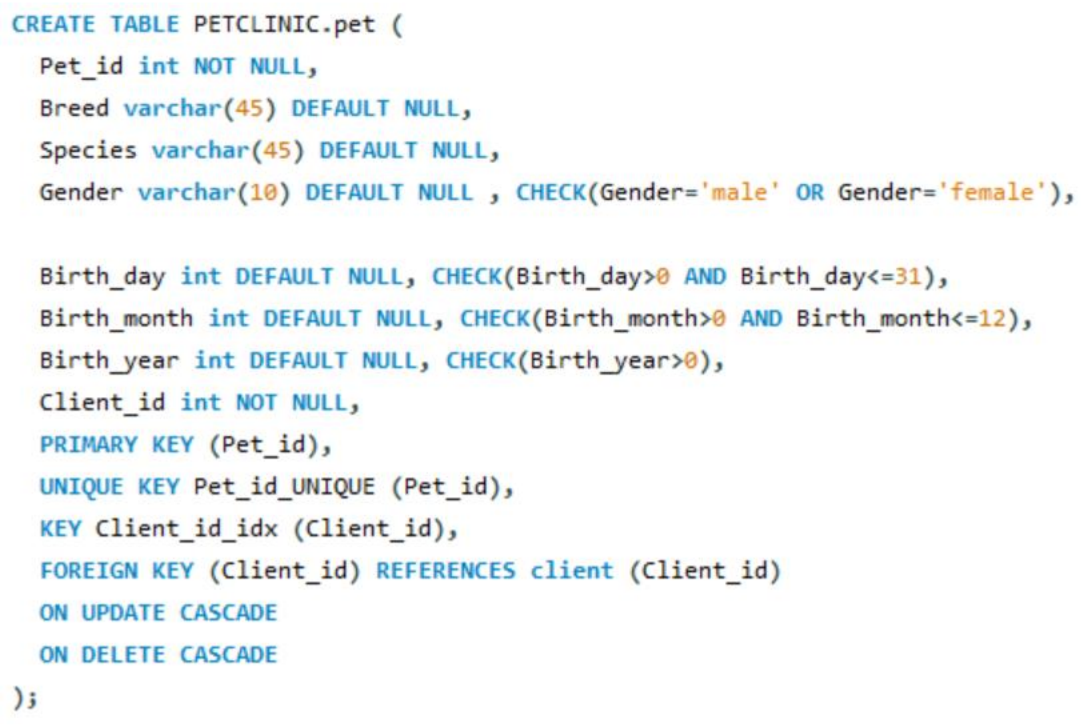
2.1 Explanation of SQL code for creating the pet tables (including constraints)

When creating the tables for the database. I used various constraints to ensure the correctness of the data being entered. For example, the primary key of pet ‘pet_id’ was specified using ‘PRIMARY KEY (Pet_id)’. The non-repetitiveness of the primary key was ensured by adding another constraint to Pet_id, using ‘UNIQUE KEY Pet_id_UNIQUE (Pet_id)’. The primary key pet_id also cannot be NULL, the constraint ‘Pet_id int NOT NULL’ was added to ensure this.

The foreign key was also specified when creating the table. For example, the foreign key client_id was set and referenced to the Client table’s primary key, using the command ‘FOREIGN KEY (Client_id) REFERENCES client (Client_id)’. The foreign key was also set as ‘NOT NULL’.

Apart from primary and foreign keys, I also used different checks to check the data inserted are valid. For example, when a gender is entered, it would check if it’s either male or female, using ‘CHECK (Gender='male' OR Gender='female')’. When entering the birthday, it would check the day entered is between 1 to 31, the month is between 1 to 12 and the year is greater than 0 etc.

Finally, I also added constraints to ensure the consistency of the database. For example, I used ‘ON UPDATE CASCADE’ and ‘ON DELETE CASCADE’ to ensure when updating and deleting, all other data related are changed.

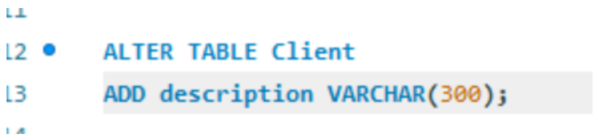


```
CREATE TABLE PETCLINIC.pet (
  Pet_id int NOT NULL,
  Breed varchar(45) DEFAULT NULL,
  Species varchar(45) DEFAULT NULL,
  Gender varchar(10) DEFAULT NULL , CHECK(Gender='male' OR Gender='female'),

  Birth_day int DEFAULT NULL, CHECK(Birth_day>0 AND Birth_day<=31),
  Birth_month int DEFAULT NULL, CHECK(Birth_month>0 AND Birth_month<=12),
  Birth_year int DEFAULT NULL, CHECK(Birth_year>0),
  Client_id int NOT NULL,
  PRIMARY KEY (Pet_id),
  UNIQUE KEY Pet_id_UNIQUE (Pet_id),
  KEY Client_id_idx (Client_id),
  FOREIGN KEY (Client_id) REFERENCES client (Client_id)
  ON UPDATE CASCADE
  ON DELETE CASCADE
);
```

Picture 2.1: code creating pet table

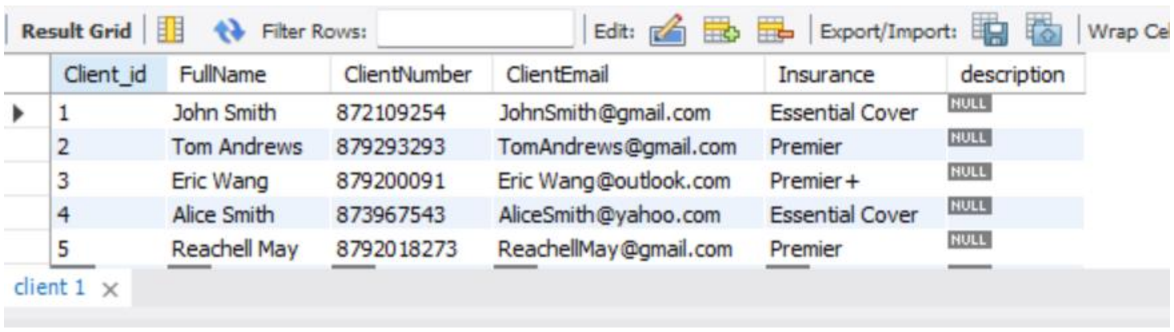
2.2 Explanation and SQL code for altering tables

In the Client's file, I created an altering table that was used to add an addition column in the client's table. I created an altering table that adds a column called "description" which holds up to 300 words. I found this might be useful because the client might want to add a text description to describe the pet etc.



```
ALTER TABLE Client
ADD description VARCHAR(300);
```

Picture 2.2: code creating altering table

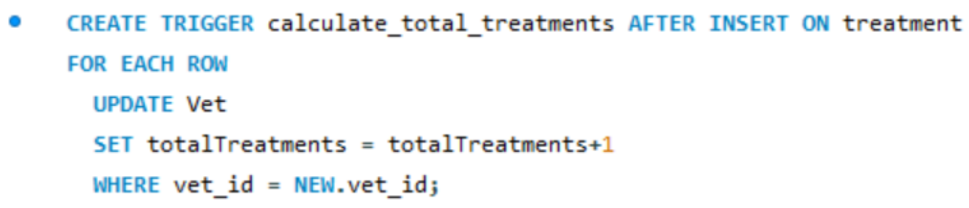


Client_id	FullName	ClientNumber	ClientEmail	Insurance	description
1	John Smith	872109254	JohnSmith@gmail.com	Essential Cover	NULL
2	Tom Andrews	879293293	TomAndrews@gmail.com	Premier	NULL
3	Eric Wang	879200091	Eric Wang@outlook.com	Premier+	NULL
4	Alice Smith	873967543	AliceSmith@yahoo.com	Essential Cover	NULL
5	Reachell May	8792018273	ReachellMay@gmail.com	Premier	NULL

Picture 2.3: code after executing altering table

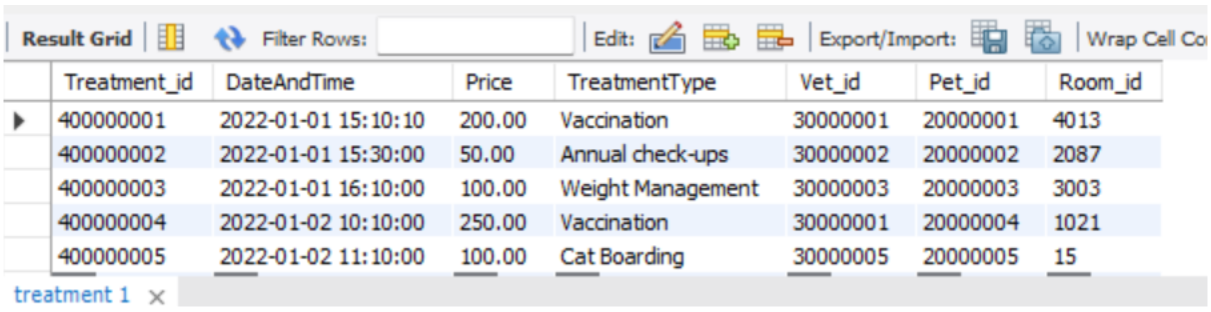
2.3 Explanation and SQL code for Trigger operations

In the treatment's file, I created a trigger that calculates the total number of treatments performed by each vet. The trigger triggers after inserting a row in the treatment's table. After inserting a row, the trigger looks at the vet_id of the new row inserted, locates the row from the vet table which has the same vet_id, then adds 1 to the total treatment of that row. For example, in picture 2.5 and 2.6, you can see that the vet with vet_id '3000001' appeared twice (performed 2 treatments), therefore, in the vet tables with vet_id '30000001' should have 2 in his total amount of treatments.



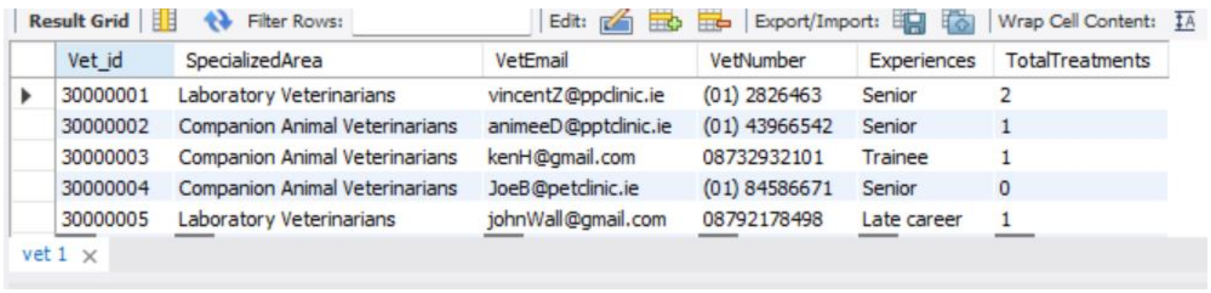
```
CREATE TRIGGER calculate_total_treatments AFTER INSERT ON treatment
FOR EACH ROW
  UPDATE Vet
  SET totalTreatments = totalTreatments+1
  WHERE vet_id = NEW.vet_id;
```

Picture 2.4: code creating trigger



Treatment_id	DateAndTime	Price	TreatmentType	Vet_id	Pet_id	Room_id
400000001	2022-01-01 15:10:10	200.00	Vaccination	30000001	20000001	4013
400000002	2022-01-01 15:30:00	50.00	Annual check-ups	30000002	20000002	2087
400000003	2022-01-01 16:10:00	100.00	Weight Management	30000003	20000003	3003
400000004	2022-01-02 10:10:00	250.00	Vaccination	30000001	20000004	1021
400000005	2022-01-02 11:10:00	100.00	Cat Boarding	30000005	20000005	15

Picture 2.5: treatments table

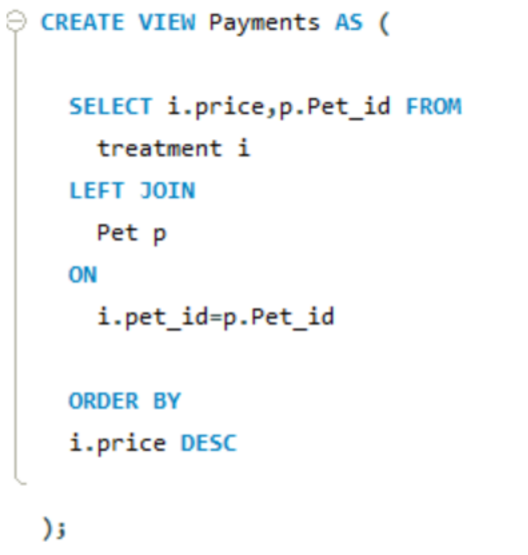


Vet_id	SpecializedArea	VetEmail	VetNumber	Experiences	TotalTreatments
30000001	Laboratory Veterinarians	vincentZ@ppclinic.ie	(01) 2826463	Senior	2
30000002	Companion Animal Veterinarians	animeeD@pptclinic.ie	(01) 43966542	Senior	1
30000003	Companion Animal Veterinarians	kenH@gmail.com	08732932101	Trainee	1
30000004	Companion Animal Veterinarians	JoeB@petclinic.ie	(01) 84586671	Senior	0
30000005	Laboratory Veterinarians	johnWall@gmail.com	08792178498	Late career	1

Picture 2.6: Vets table

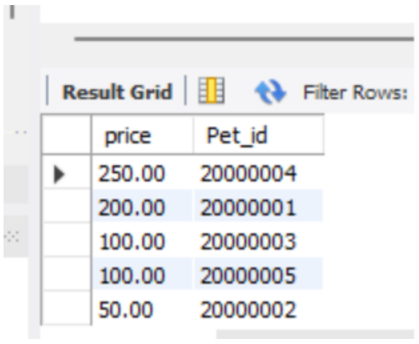
2.4 Explanation and SQL code for Creation of Views

In the treatments file, I created a view called payments that shows all the payments for each pet in a descending order. (a table containing price, pet_id). I believe this view helps to keep track of the clinic's income.



```
CREATE VIEW Payments AS (

  SELECT i.price,p.Pet_id FROM
    treatment i
  LEFT JOIN
    Pet p
  ON
    i.pet_id=p.Pet_id

  ORDER BY
  i.price DESC

);
```

Picture 2.7: code creating view

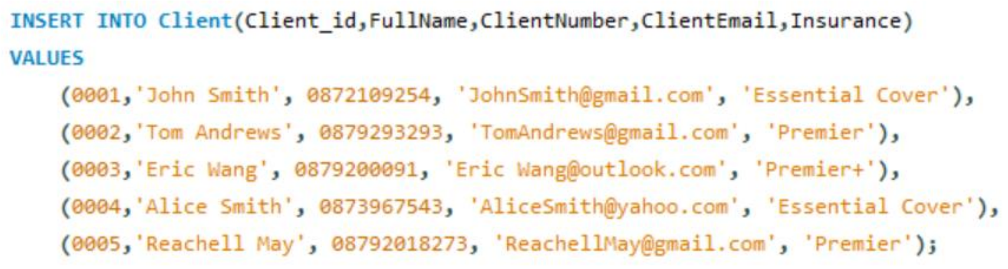


price	Pet_id
250.00	20000004
200.00	20000001
100.00	20000003
100.00	20000005
50.00	20000002

Picture 2.8: view tables

2.5 Explanation and SQL code for one of the commands to Populate the Tables

The Tables were populated by the 'INSERT INTO' function and the data were passed in as "VALUES". For example, the Client's table shown in Picture 2.3 was populated by inserting the client's id, full name, client's number, client's email and insurance one by one respectively into the table.



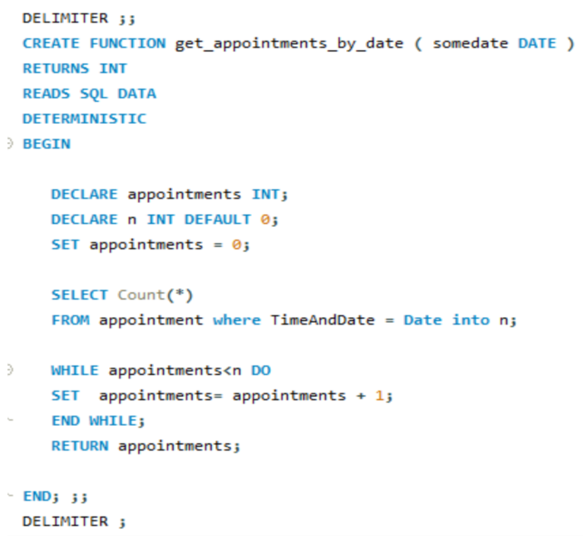
```
INSERT INTO Client(Client_id,FullName,ClientNumber,ClientEmail,Insurance)
VALUES
    (0001,'John Smith', 0872109254, 'JohnSmith@gmail.com', 'Essential Cover'),
    (0002,'Tom Andrews', 0879293293, 'TomAndrews@gmail.com', 'Premier'),
    (0003,'Eric Wang', 0879200091, 'Eric Wang@outlook.com', 'Premier+'),
    (0004,'Alice Smith', 0873967543, 'AliceSmith@yahoo.com', 'Essential Cover'),
    (0005,'Reachell May', 08792018273, 'ReachellMay@gmail.com', 'Premier');
```

Picture 2.9: code for populating the client's table

2.6 Explanation and example SQL Code for retrieving information from the database (including use of Joins and use of functions)

In order to help retrieving information from the database, I used various techniques, including left joins, functions and procedure language (PL). For example, in the appointments file, I created a function that calculates the total appointments with a certain date (takes a DATE type as input and returns an integer type). The implementation is in PL, for example, I declared a variable called appointments which act as a counter. I believe this function will be helpful for the vets, for example, if they are too many appointments with a certain date, the vets should reorganise treatment times.

I also used left join in the treatments file to create the view shown in picture 2.7 and 2.8. It joins the price with the pet_id which helps to track all the payments.



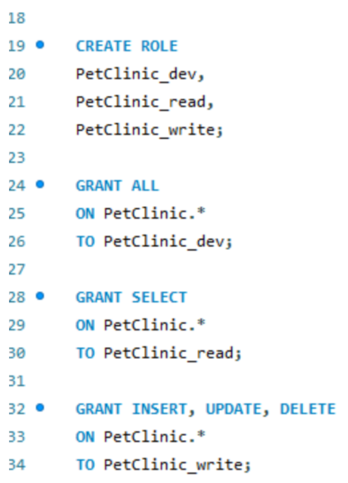
```
DELIMITER ;;
CREATE FUNCTION get_appointments_by_date ( somedate DATE )
RETURNS INT
READS SQL DATA
DETERMINISTIC
BEGIN

    DECLARE appointments INT;
    DECLARE n INT DEFAULT 0;
    SET appointments = 0;

    SELECT Count(*)
    FROM appointment where TimeAndDate = Date into n;

    WHILE appointments<n DO
    SET  appointments= appointments + 1;
    END WHILE;
    RETURN appointments;

END; ;;
DELIMITER ;
```

Picture 3.0: code for retrieving data using functions/PL

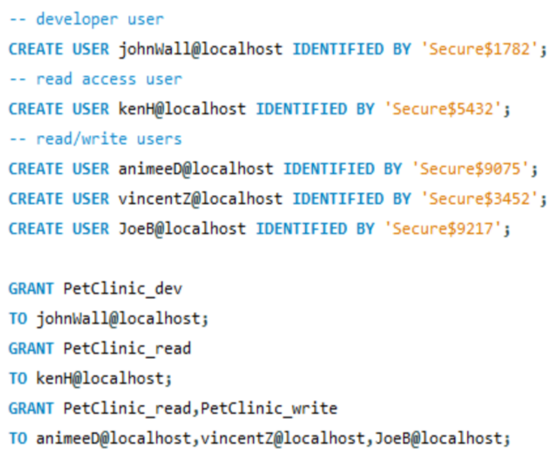
2.7 Explanation and SQL Code for any Security commands (roles & permissions)

In the vet’s file, I created three roles: developing (PetClinic_dev), reading (PetClinic_read) and writing (PetClinic_write). The developing role was granted all permissions, it is for the manager of the database. The writing role was granted permissions to insert, delete and update the tables. The reading role was only grated permissions to make do selects on the tables.

Johnathan Wall is in charge of the company; therefore, his account has all privileges (assigned with the developing role). Since there are no staff in the clinic, the senior veterans Animee, Vincent and Joe have the permission to edit the tables (assigned with the writing role). However, ken which is a trainee should have less privilege and not allowed to change the tables, therefore assigned with the reading role.



```
CREATE ROLE
PetClinic_dev,
PetClinic_read,
PetClinic_write;

GRANT ALL
ON PetClinic.*
TO PetClinic_dev;

GRANT SELECT
ON PetClinic.*
TO PetClinic_read;

GRANT INSERT, UPDATE, DELETE
ON PetClinic.*
TO PetClinic_write;
```



```
-- developer user
CREATE USER johnWall@localhost IDENTIFIED BY 'Secure$1782';
-- read access user
CREATE USER kenH@localhost IDENTIFIED BY 'Secure$5432';
-- read/write users
CREATE USER animeeD@localhost IDENTIFIED BY 'Secure$9075';
CREATE USER vincentZ@localhost IDENTIFIED BY 'Secure$3452';
CREATE USER JoeB@localhost IDENTIFIED BY 'Secure$9217';

GRANT PetClinic_dev
TO johnWall@localhost;
GRANT PetClinic_read
TO kenH@localhost;
GRANT PetClinic_read,PetClinic_write
TO animeeD@localhost,vincentZ@localhost,JoeB@localhost;
```

Picture 3.1&3.2: code for security commands (roles/permissions)

3 Listing of SQL Code for the database

Appointment.SQL
Books.SQL
Client.SQL
Pet.SQL
Room.SQL
Sends.SQL
ServiceType.SQL
Treatment.SQL
Vet.SQL

(Code are submitted as separate files)