3. Translate the logical data model for the Oracle Enterprise DBMS. (12/08/22)

a. Develop SQL code to create the entire database schema, reflecting the constraints identified in previous steps.

Using the example code in connect_sqlite.py, I made my own file main.py which abstracts the sqlite3 calls into a database class. This makes the code a lot cleaner, since in the main function I can write a single line such as

```
db = database('pawsome_pets.db')
```

and the constructor of the database class will make the sqlite3 database with the given name if it does not already exist, save a connection and cursor object for later use, and execute all of the queries in init_queries, which is a list of CREATE TABLE queries in order to make all the necessary tables each with their primary keys, foreign keys, and integrity constraints described in part two. I also have a destructor for the database class, called automatically when the object goes out of scope or the program ends, which commits the changes and closes the connection. After instantiating the database class, we can print out all of our tables using a SELECT query in order to make sure nothing went wrong:

```
# instantiate database object (constructor will initalize it with init_queries)
db = database('pawsome_pets.db')

# get all tables to make sure init_queries succeeded
db.exec(['''SELECT name FROM sqlite_master WHERE type = 'table';'''])
```

And we can see that all of our tables were created successfully when the exec function prints the results:

```
C:\Users\Myles\Desktop\part3>python main.py
[('Staff',), ('Examination',), ('Clinic',), ('ClinicLocation',), ('Pet',), ('Owner',), ('Zip',)]
```

One thing I decided to change from part two is that, instead of INT, I made the telephoneNo attributes VARCHAR(15) and the zip attribute VARCHAR(10). The reason for this is that telephone numbers and zip codes contain or could contain dashes, and you can only store numerical values with the INT data type, so the string-like data type VARCHAR was much more suitable in this situation. I also decided to make the DATE data types into VARCHAR(10) since SQLite doesn't have a built-in datetime type (they will use the format DAY/MONTH/YEAR ex. '02-15-1985'). Additionally, for the integer primary keys I didn't need to specify NOT NULL or UNIQUE or AUTOINCREMENT, since by default in SQLite if you specify an INTEGER PRIMARY KEY it will not be nullable and auto increment by default. Finally, we want to make sure we add UNIQUE (fName, lName) to Staff, Pet, and Owner since we may only have this information and not their ID and we want to be able to run queries such as the ones in part c with the expectation that we will get one row back (for example, if an owner gives us a pet's first and last name, we want to get only the information about their pet and not someone else's pet that may have an identical name if we didn't use UNIQUE).

I wanted the code to be as readable as possible, and so I thought it made sense to make init_queries a global variable at the top of the program in order to distinguish it from the queries in the main function, since init_queries only contains initialization queries and the other queries in the program will be those acting on an already initialized database (adding records to tables, performing operations on them, etc.).

b. Create at least 5 tuples for each relation in your database.

In the main function, I have a variable called sample_data_queries which is a list of all the INSERT INTO queries for inserting 5 tuples of sample data per table. The Pet table, for example, is populated with data by these queries:

```
# 5 sample rows to insert into Pet table
'''INSERT INTO Pet (fName, lName, DOB, species, breed, color, clinicNo, ownerNo) VALUES ('Max', 'Rogers', '04-24-2019', 'canine', 'German Shepherd', 'brown', 3, 2)''',
'''INSERT INTO Pet (fName, lName, DOB, species, breed, color, clinicNo, ownerNo) VALUES ('Moonpie', 'Pimentel', '05-31-2021', 'feline', 'Domestic Shorthair', 'black/white', 5, 1)''',
'''INSERT INTO Pet (fName, lName, DOB, species, breed, color, clinicNo, ownerNo) VALUES ('Lily', 'Blanche', '02-13-2022', 'rabbit', 'New Zealand White', 'white', 4, 4)''',
'''INSERT INTO Pet (fName, lName, DOB, species, breed, color, clinicNo, ownerNo) VALUES ('Hershey', 'Perry', '11-04-2018', 'canine', 'Labrador Retriever', 'brown', 1, 3)''',
'''INSERT INTO Pet (fName, lName, DOB, species, breed, color, clinicNo, ownerNo) VALUES ('Chip', 'Ehrmantraut', '07-21-2017', 'bird', 'Parakeet', 'green/yellow', 2, 2)''',
```

After running db.exec on sample_data_queries, we can then print out the result of running SELECT * on every table in order to verify that they all have 5 new records:

```
C:\Users\Myles\Desktop\part3>python main.py
[('Staff',), ('Examination',), ('Clinic',), ('ClinicLocation',), ('Pet',), ('Owner',), ('Zip',)]

[(1, 'John', 'Smith', 'Chestnut Ave', 'Birmingham', '124-994-2293', '02-15-1985', 'Secretary', 55000, 1), (2, 'Ashley', 'Graham', 'Fawn St', 'Westport', '305-724-6923', '02-16-2001', 'Veterinarian', 85000
, 2), (3, 'Leon', 'Kennedy', 'Rose Blv', 'Fall River', '101-444-8792', '10-01-1990', 'Janitor', 30000, 1), (4, 'Patricia', 'Hoxforn', 'Highland Ave', 'Dartmouth', '774-231-4012', '07-12-1998', 'Veterinary
 Technician', 68000, 4), (5, 'Alexander', 'Rutherford', 'Waning Crescant Dr', 'Cambridge', '808-156-6832', '05-19-1993', 'Marketing Specialist', 70000, 3)]

[(1, 'Broken leg', 'Canine Max fractured leg while running', '08-15-2022', 'Fitted with doggie cast', 3, 1), (2, 'Scratched cornea', 'Feline Moonpie was scratched in the eye by his brother while playing',
 '10-27-2022', 'Prescribed eye drops', 2, 5), (3, 'Ear mites', 'Rabbit Lily has severe ear mite infestation', '12-01-2022', 'Prescribed antibiotics', 4, 4), (4, 'Kennel Cough', 'Canine Hershey contracted
Kennel Cough after going to daycare without getting vaccinated first', '04-13-2022', 'Told to rest, get vaccinated, and stay away from daycare for now', 1, 2), (5, 'Broken wing', 'Bird Chip broke wing whi
le flying indoors', '09-05-2022', 'Fitted with wing cast', 2, 1)]

[(1, 'Birmingham Pawsome Pets'), (2, 'Westport Pawsome Pets'), (3, 'Fall River Pawsome Pets'), (4, 'Dartmouth Pawsome Pets'), (5, 'Cambridge Pawsome Pets')]

[('Birmingham Pawsome Pets', 'Picnic St', 'Birmingham', '705-724-4552'), ('Westport Pawsome Pets', 'Auburn Dr', 'Westport', '902-611-0951'), ('Fall River Pawsome Pets', 'Jeffrey St', 'Fall River', '508-91
1-6933'), ('Dartmouth Pawsome Pets', 'Palace Ave', 'Dartmouth', '774-811-1329'), ('Cambridge Pawsome Pets', 'West Rodney St', 'Cambridge', '202-113-0009')]

[(1, 'Max', 'Rogers', '04-24-2019', 'canine', 'German Shepherd', 'brown', 3, 2), (2, 'Moonpie', 'Pimentel', '05-31-2021', 'feline', 'Domestic Shorthair', 'black/white', 5, 1), (3, 'Lily', 'Blanche', '02-1
3-2022', 'rabbit', 'New Zealand White', 'white', 4, 4), (4, 'Hershey', 'Perry', '11-04-2018', 'canine', 'Labrador Retriever', 'brown', 1, 3), (5, 'Chip', 'Ehrmantraut', '07-21-2017', 'bird', 'Parakeet', '
green/yellow', 2, 2)]

[(1, 'Jamie', 'Rogers', 'Rockland Ave', 'Birmingham', '643-102-5992'), (2, 'Jackson', 'Pimentel', 'Pickle Dr', 'Westport', '508-291-3312'), (3, 'Irene', 'Blanche', 'Peanut St', 'Fall River', '330-500-2235
'), (4, 'Cherry', 'Perry', 'Mallory St', 'Dartmouth', '808-104-4492'), (5, 'Monica', 'Ehrmantraut', 'Leaf Rd', 'Cambridge', '774-001-0192')]

[('Birmingham', '61021'), ('Westport', '2790'), ('Fall River', '99526'), ('Dartmouth', '83321'), ('Cambridge', '10923')]
```

c. Develop 5 SQL queries using embedded SQL (see Python tutorial).

We will make another variable in main, example_queries, which will be our list of 5 SQL queries in order to demonstrate the use of the database.

1) If a pet owner calls the clinic and wants a complete history of examinations done on their pet, we will need a query which can retrieve this information for us given the first and last name of a pet. Let's say we want to get all examinations done on the cat Moonpie Pimentel; we will select the pet's petNo from the Pet table using his first and last name, and then select all rows from the Examinations table which have this petNo. This is the query and what it returned:

```
'''SELECT * FROM Examination WHERE petNo =
      (SELECT petNo FROM Pet WHERE fName = 'Moonpie' AND lName = 'Pimentel')'''
```

```
[(2, 'Scratched cornea', 'Feline Moonpie was scratched in the eye by his brother while playing', '10-27-2022', 'Prescribed eye drops', 4, 2)]
```

We can see that it worked since it gave us the only examination in our database for Moonpie.

2) Let's say instead of the full name of the pet, we have the full name of a pet owner, and we want to find all examinations done on all of this pet owner's pets. This is a little more complicated, since the Examination table does not store ownerNo. It does store petNo though, so now we know we need to get all petNo's belonging to a certain ownerNo from the Pet table. To get ownerNo, we query Owner with the first and last name. So we will be querying 3 different tables to do this. We can tell from our Pet table that Moonpie and Lily are both owned by the owner with ownerNo 1, Jamie Rogers. Let's say Jamie comes into the clinic, gives his full name, and wants all examination information on all his pets. This is our query and what it returns:

```
'''SELECT * FROM Examination WHERE petNo IN
      (SELECT petNo FROM Pet WHERE ownerNo =
          (SELECT ownerNo FROM Owner WHERE fname = 'Jamie' AND lName = 'Rogers'))'''
```

```
[(2, 'Scratched cornea', 'Feline Moonpie was scratched in the eye by his brother while playing', '10-27-2022', 'Prescribed eye drops', 4, 2), (3, 'Ear mites', 'Rabbit Lily
has severe ear mite infestation', '12-01-2022', 'Prescribed antibiotics', 3, 3)]
```

We can see it worked since we got back both Moonpie and Lily's examination records.

3) Instead of information about pet examinations, we might want to get some information about employees. Let's say we want to get the average salary of all employees that work at the Birmingham clinic. We can see that John Smith and Leon Kennedy both work at this clinic, and the average of their salaries is (55000 + 30000) / 2 = 42500, so this is what we should get.

```
'''SELECT AVG(salary) FROM Staff WHERE staffNo IN
      (SELECT staffNo FROM Staff WHERE clinicNo IN
          (SELECT clinicNo FROM Clinic WHERE name IN
              (SELECT name FROM ClinicLocation WHERE city = 'Birmingham')))'''
```

```
[(42500.0,)]
```

Since we got 42500, we know it worked.

4) We might also want to get an idea of how busy our clinics are based on how many pets we have registered at a particular location. We can see based on the Pet table that both Max the dog and Chip the bird are registered at clinicNo 3 (the Fall River clinic), and all other clinics have just 1 registered pet, so let's develop a query to count the number of registered pets at a certain location and make sure the numbers we get back are correct.

```
'''SELECT COUNT(*) FROM Pet WHERE clinicNo IN
      (SELECT clinicNo FROM Clinic WHERE name IN
          (SELECT name FROM ClinicLocation WHERE city = 'Fall River'))'''
```

When we run this query with 'Fall River' as the city, we get 2 because there are 2 pets registered at the Fall River location:

```
[(2,)]
```

But if we run it with a different city, like Dartmouth:

```
'''SELECT COUNT(*) FROM Pet WHERE clinicNo IN
      (SELECT clinicNo FROM Clinic WHERE name IN
          (SELECT name FROM ClinicLocation WHERE city = 'Dartmouth'))'''
```

We can see that we get 1:

```
[(1,)]
```

So this query is working correctly. This would be a very useful query if we had many locations and many customers and we want to see which locations are the busiest so maybe we can open more clinics in that area to lessen the load.

5) Finally, perhaps we suspect there is some sort of trend of sickness or injury among dogs in our area, so we want to get all examination information about exams that were performed on canines. We will get all canine petNo's from our Pet table, and then get the 'complaint' attribute from the Examination table for all records that have these petNo's:

```
'''SELECT complaint FROM Examination WHERE petNo IN
      (SELECT petNo FROM Pet WHERE species = 'canine')'''
```

This query yields:

```
[('Broken leg',), ('Kennel Cough',)]
```

Which makes sense since we have two canines in our database, Max who broke his leg and Hershey who got Kennel Cough. We could take this a step further and get all information on canines with broken bones using the LIKE operator:

```
'''SELECT complaint FROM Examination WHERE petNo IN
      (SELECT petNo FROM Pet WHERE species = 'canine') AND (complaint LIKE '%Broke%' OR complaint LIKE '%broke%') '''
```

And this will get us all exam complaints for dogs which contains the string 'Broke' or 'broke', which is just Max:

```
[('Broken leg',)]
```

But we could imagine this query would be very useful if we had a large number of examination records and we want to see which sicknesses or injuries are most common.

d. Upload all the code and documentation to GitHub.

https://github.com/msg203/pawsomepets