Second Chance Animal Shelter

CS 3200: Database Design

Professor Kathleen Durant

Namleen Kohli, Kyra Grimes, Luke Colombo

README

We used MySQL to build the backend and Python to build the frontend. To connect the backend and frontend, we used pymysql. You will need to have all 3 of these to run our program. The following are links to download the software.

1. MySQL: https://dev.mysql.com/downloads/installer/
2. Python: https://www.python.org/downloads/
3. pymysql: https://pypi.org/project/pymysql/

After installing MySQL server, you will need to create the database on your local machine using the three SQL files provided. The backend of our application consists of three SQL files, `animal_shelter.sql`, `insert_data.sql`, and `procedures.sql`. To initialize the database, run `animal_shelter.sql` to create the tables, then `insert_data.sql` to insert the records, and then `procedures.sql` to create the necessary triggers and procedures. Alternatively, you can run the db_dump file by doing Server → Data Import in Workbench.

The folder `database-front-end` contains all the necessary code for the front-end application. You can run `frontend.py` entirely on the command line (`python3 frontend.py`). It should first prompt you for a username and password, this should be 'root' for the username and your password for MySQL server. After that, you are connected to the MySQL database and can run the application as intended.

There are four different views for our database: new visitor, returning visitor, staff, and manager. The only operation new visitors can do is enter in personal information in order to create an account. For the three other views, after selecting a certain one, the application will print a list of actions you can do.

Technical Specifications

We used MySQL, via the Workbench IDE, to build the backend database, including the tables, procedures, and triggers. The frontend is built entirely in Python, using pymysql to connect to the

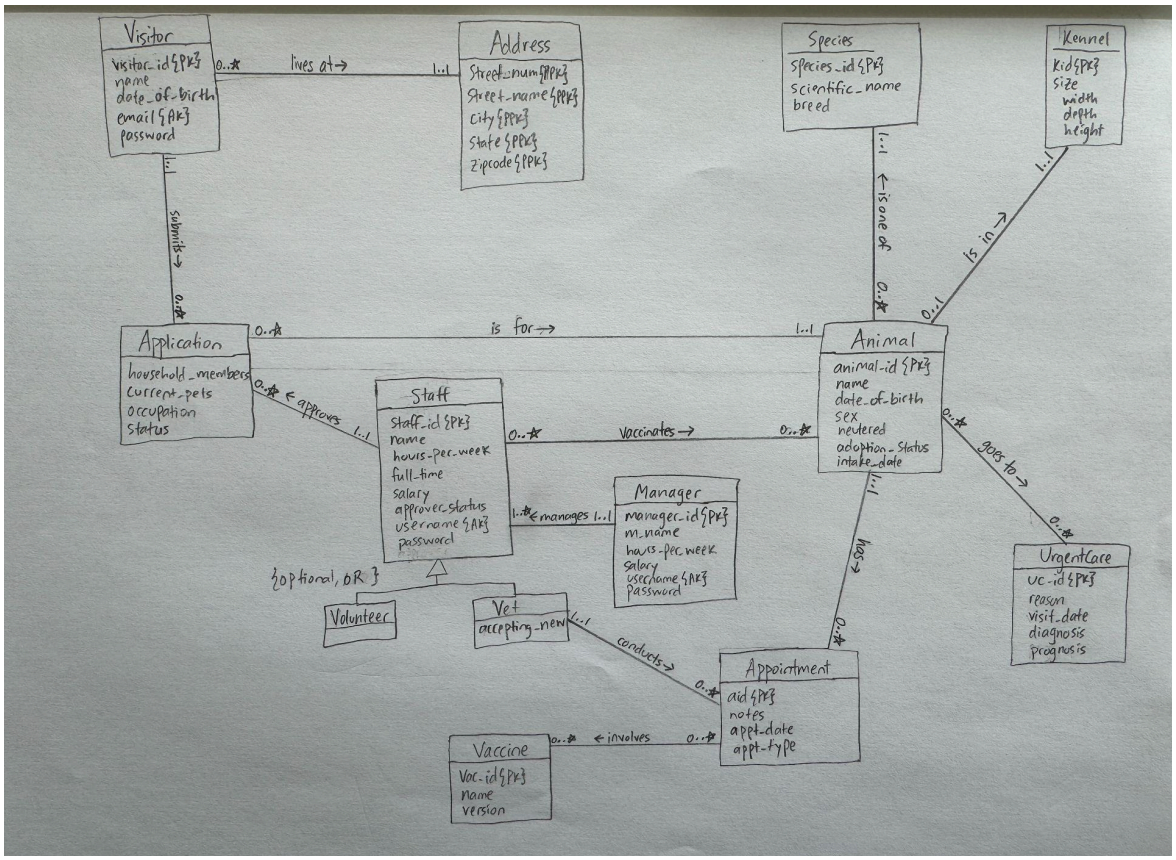
backend. With Python, we imported datetime to convert string input in Python to a date type in MySQL.

Textual Description

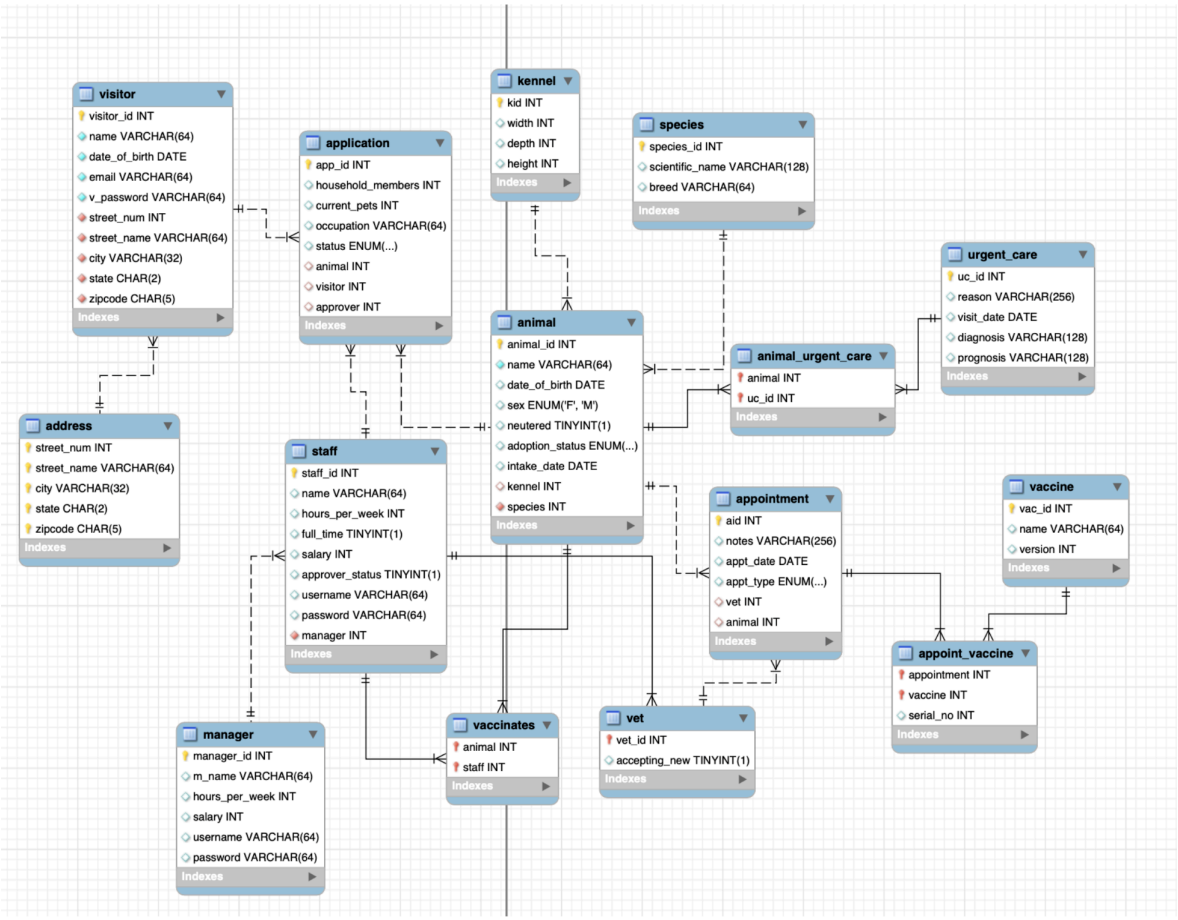
The animal shelter has many entities and relationships representing different users of the shelter and the corresponding actions they can take. The shelter has animals. An animal has an ID, a name, a date of birth, a sex (either M or F), if they have been neutered (T or F), an adoption status (foster, shelter, or adopted), and an intake date. Animals live in a kennel, which has an ID, a size made up of the width, depth, and height. An animal is also a specific species. A species has an ID, a scientific name, and a breed. An animal might also need to go to urgent care. Urgent care requires an ID, a reason for the trip, a visit date, a diagnosis, and a prognosis. An animal could also have an appointment for a check up or a vaccination. An appointment consists of an ID, notes, an appointment date, and an appointment type (checkup or vaccination). If an appointment is of type vaccination, then it would require a vaccine. Vaccines consist of an ID, name, and a version. To run the shelter, there are managers and staff. Managers have an ID, a name, a count of how many hours they work per week, a salary, a unique username, and a password. Staff also have an ID, a name, a count of how many hours they work per week, if they are full time (T or F), a salary, if they have approver status (T or F), a unique username, and a password. There are two types of staff, a volunteer or a vet. A vet as an additional attribute, if they are accepting new patients (T or F). The shelter has a visitor, who has a unique ID, a name, a date of birth, a unique email address, and a password. A visitor lives at a specific address. An address has a street number, street name, city, state, and zip code. A visitor can submit an application to adopt an animal from the shelter. An application requires a count of household members, a count of current pets, an occupation, and a status (pending, accepted, or denied).

An animal must be in only 1 kennel and a kennel can have at most 1 animal, unless it is empty. Also, an animal is one of only 1 species but a species can have either no animals or many animals. If an animal needs to, they can go on many trips to urgent care and an urgent care can have many animals. An appointment can only be for 1 animal but an animal can have many appointments. If the appointment is a vaccination, it can involve many vaccines and a vaccine can be used at many appointments. To conduct the appointments, we need employees. A manager manages 1 to many staff members but a staff member has only 1 manager. A staff member can be either a volunteer or a vet. Staff can vaccinate many animals and an animal can be vaccinated by many staff members. However, only a vet can conduct an appointment. Vets can conduct many appointments but an appointment can only be conducted by 1 vet. To adopt these animals, we have visitors who submit applications. Firstly, a visitor lives at only 1 address but an address can house many visitors. A visitor can submit many applications to adopt an animal but an application can only be submitted by 1 visitor. An application is only for 1 animal but many applications can be for the same animal. Lastly, an application is approved by only 1 staff member but a staff member can approve multiple applications.

UML Diagram


Visitor
visitor_id {PK}
name
date_of_birth
email {AK}
password
lives at
Address
Street_num{PPK}
Street_name{PPK}
City{PPK}
State{PPK}
Zipcode{PPK}
Species
species_id {PK}
scientific_name
breed
Kennel
kid{PK}
size
width
depth
height
submits
is one of
is in
Application
household_members
current_pets
occupation
status
is for
Animal
animal_id {PK}
name
date_of_birth
sex
neutered
adoption_status
intake_date
approves
Staff
staff_id {PK}
name
hours_per_week
full_time
salary
approver_status
username {AK}
password
vaccinates
goes to
Manager
manager_id {PK}
m_name
hours_per_week
salary
username {AK}
password
manages
{optional, OR}
Volunteer
Vet
accepting_new
conducts
has
UrgentCare
uc_id {PK}
reason
visit_date
diagnosis
prognosis
Appointment
aid {PK}
notes
appt_date
appt_type
Vaccine
Vac_id {PK}
name
version
involves

Logical design



Final User Flow

To start the application, the user must enter their credentials to connect to the backend database. The following user flows are assuming the user has connected to the backend.

Staff view: (a staff member who works at the animal shelter)

1. Type ‘staff’ for the type of user
2. Enter a valid staff username
3. Enter a valid staff password
4. See all the possible staff actions
5. Choose an action to do by typing the shorthand of the action, provided in parenthesis
   See shelter animals action

1. Type ‘see_shelter_animals’
2. See the animals in the shelter

Look up animal action

1. Type ‘look_up_animal’
2. Enter the animal’s name (if not known, hit ‘Enter’)
3. Enter the animal’s ID (if not known, hit ‘Enter’)
4. See the results

See app status action

1. Type ‘see_app_status’
2. Enter the visitor’s email address
3. See the results

See stats action

1. Type ‘see_stats’
2. See the statistics of the animal shelter

Add new animal action

1. Type ‘add_new_animal’
2. Enter the animal’s name
3. Enter the animal’s date of birth
4. Enter the animal’s sex (M/F)
5. Enter if the animal’s been neutered (T/F)
6. Enter the animal’s intake date
7. Enter the animal’s kennel ID
8. Enter the animal’s species
9. Enter the animal’s breed
10. See the results

Make appt action

1. Type ‘make_appt’
2. Enter the type of appointment (check up/vaccination)
3. If check up:
    a. Enter the appointment notes
    b. Enter the appointment date
    c. Enter the appointment vet’s ID
    d. Enter the appointment animal’s ID
4. If vaccination:
    a. Enter the appointment notes
    b. Enter the appointment date
    c. Enter the appointment vet’s ID
    d. Enter the appointment animal’s ID
    e. Enter the appointment vaccine name
    f. Enter the appointment vaccine version

        g. Enter the appointment vaccine serial number
    5. See the results

Manager view: (a manager who works at the animal shelter and manages staff members)

1. Type manager for the type of user
2. Enter a valid manager username
3. Enter a valid manager password
4. See all the possible manager actions
5. Choose an action to do by typing the shorthand of the action, provided in parenthesis

    <u>See shelter animals action</u>
        1. Type ‘see_shelter_animals’
        2. See the animals in the shelter

    <u>Look up animal action</u>
        1. Type ‘look_up_animal’
        2. Enter the animal’s name (if not known, hit ‘Enter’)
        3. Enter the animal’s ID (if not known, hit ‘Enter’)
        4. See the results

    <u>See app status action</u>
        1. Type ‘see_app_status’
        2. Enter the visitor’s email address
        3. See the results

    <u>See stats action</u>
        1. Type ‘see_stats’
        2. See the statistics of the animal shelter

    <u>Add new animal action</u>
        1. Type ‘add_new_animal’
        2. Enter the animal’s name
        3. Enter the animal’s date of birth
        4. Enter the animal’s sex (M/F)
        5. Enter if the animal’s been neutered (T/F)
        6. Enter the animal’s intake date
        7. Enter the animal’s kennel ID
        8. Enter the animal’s species
        9. Enter the animal’s breed
        10. See the results

    <u>Make appt action</u>
        1. Type ‘make_appt’
        2. Enter the type of appointment (check up/vaccination)
        3. If check up:
            a. Enter the appointment notes
            b. Enter the appointment date

        c. Enter the appointment vet's ID
        d. Enter the appointment animal's ID
    4. If vaccination:
        a. Enter the appointment notes
        b. Enter the appointment date
        c. Enter the appointment vet's ID
        d. Enter the appointment animal's ID
        e. Enter the appointment vaccine name
        f. Enter the appointment vaccine version
        g. Enter the appointment vaccine serial number
    5. See the results

Remove staff action

1. Type 'remove_staff'
2. Enter the staff to be removed's ID
3. See the results

Remove animal action

1. Type 'remove_animal'
2. Enter the animal to be removed's ID
3. See the results

Add a staff action

1. Type 'add_staff'
2. Enter the staff's name
3. Enter the staff's hours per week to be worked
4. Enter if the staff's full time (T/F)
5. Enter the staff's salary
6. Enter the staff's approver status (T/F)
7. Enter the staff's username
8. Enter the staff's password
9. Enter the staff's manager's ID
10. See the results

New visitor view: (a new visitor to the animal shelter, does not have an username and password yet)

1. Type 'new' for the type of user
2. Follow the prompt to create an account (username and password)
    a. Enter a name for the new visitor
    b. Enter a date of birth for the new visitor
    c. Enter an email address for the new visitor
    d. Enter a password for the new visitor
    e. Enter an address for the new visitor
        i. Enter a street number

        ii. Enter a street name
        iii. Enter a city
        iv. Enter a state
        v. Enter a zipcode
3. See the results

Returning visitor view: (a returning visitor to the animal shelter)

1. Type ‘returning’ for the type of user
2. Enter a valid returning visitor username
3. Enter a valid returning visitor password
4. See all possible returning visitor actions

   Animals with no app action
   1. Type ‘animals_with_no_app’
   2. See the results

   Submit app action
   1. Type ‘submit_app’
   2. Enter a valid visitor email address
   3. Enter the count of the visitor’s household members
   4. Enter the count of the visitor’s current pets
   5. Enter the visitor’s occupation
   6. Enter the animal’s ID for whom the application is for
   7. See the results

   Update address action
   1. Type ‘update_address’
   2. Enter a valid visitor’s email address
   3. Enter the updated address
      a. Enter the updated street number
      b. Enter the updated street name
      c. Enter the updated city
      d. Enter the updated state
      e. Enter the updated zipcode
   4. See the results

Noted Bugs

One minor bug we would like to fix is in the *update_address* function in *frontend.py*. At the moment, a user can enter in any email and update that person’s address; we would like to add a check to ensure they’re updating their own email, not anyone else’s.

Lessons Learned

Technical expertise gained

Through this project, we learned how to connect a backend SQL database to the frontend of an application. We learned how to use pymysql, specifically a cursor and connection that runs the procedures in the backend and updates the tables in the database, respectively. We gained lots of experience with the create and select statements in SQL, as well as with writing procedural code in SQL. Additionally, we honed our skills in creating Python command-line applications.

Insights

During this project, we managed our time well because we had a game plan. We started with the SQL backend, creating tables, adding data, and then implementing triggers and procedures. Once we had a stable and functioning backend, we implemented the frontend. However, while building the frontend, we had to revise the procedures in the backend for error handling. Despite this minor setback, we were able to finish our project in a timely manner.

An insight we had into the data domain was when we accessed the procedure from the frontend via cursor.callproc(). Many of our procedures use date or integer data types but the input from Python is always a string. At first, we thought about changing the data types in the database to be all varchar but ultimately we decided against this to ensure the versatility of the database. So, we had to ensure the user entered the correct data type in the correct format (for dates). This process compelled us to review our data domains.

Alternative design/approaches to project

During our design, we considered multiple approaches to the staff/manager relationship. Originally, Manager was a subclass of Staff, with a recursive relationship (one Manager manages many Staff). However, this recursion was difficult to represent in the logical design, so we made Manager its own table, at the cost of some code duplication with its attributes. We also went through various designs for vaccinations—originally, vaccines were a relationship attribute between staff and animal, then we added a table to keep track of the vaccinations in a specific veterinarian visit. Additionally, we used to have an entity for an intake form, representing a piece of paperwork that an employee would fill out to officially add an animal to the database. However, we scrapped this idea in order to simplify our Staff/Animal relationships.

Future Work

Planned uses of database

We do not plan on using this database in the future.

Potential areas for added functionality

We would add more specific error messaging on the backend, to more precisely capture why errors are happening. We are satisfied with our error handling, but our messages could be more user-friendly or descriptive. Additionally, we had more ideas for CRUD operations that we did not end up implementing for the sake of time, including: withdrawing an application, updating the approver on an application, and changing your password (where you would have to enter your old password first to confirm).