DBMS

Final Project

Group 24

0716089 王柏舜

0816004 吳原博

0816102 陳品戎

0816203 陳永諭

Data

Data

1. 動物認領養

Contents

This file contains animals kept by the public shelters in Taiwan. One can find the basic information of the animals and the shelters they are staying in.

Source

行政院農業委員會資料開放平台 (coa.gov.tw)

Format

A csv file which can be downloaded directly.

Rows

9999.

Columns

Name	Meaning	Format
animal_id	As the name suggests. Primary key to the data.	Integer
animal_subid	The id for the location of the animal.	A string of digits, alphabets and "-"
animal_area_pkid	The id for the province where the animal is.	Integer
animal_shelter_pkid	The id of the shelter where the animal is.	Integer
animal_place	Where the animal is.	String
animal_kind	As the name suggests.	{ 狗, 貓 }
animal_sex	As the name suggests.	{ F (female), M (male), N (unknown) }
animal_bodytype	The size of the animal.	{ SMALL, MEDIUM, BIG }
animal_colour	As the name suggests.	String

animal_age	As the name suggests.	{ ADULT, CHILD }
animal_sterilization	Whether the animal is neutere d.	{ T (true), F (false) }
animal_bacterin	Whether the animal has inject ed a rabies vaccine.	{ T (true), F (false) }
animal_foundplace	As the name suggests.	String
animal_title	Title of the animal.	Unknown. This column is em pty in all rows.
animal_status	As the name suggests.	{ OPEN, ADOPTED, DEAD, NONE, OTHER }
animal_remark	As the name suggests.	String
animal_caption	Other information.	Unknown. This column is em pty in all rows.
animal_opendate	The date from which the anim al is open for adoption.	yyyy-mm-dd
animal_closeddate	The date from which the anim al is closed for adoption.	yyyy-mm-dd
animal_update	The date of the previous updat e for the animal.	yyyy/mm/dd
animal_createtime	As the name suggests.	yyyy/mm/dd
shelter_name	As the name suggests.	String
ablum_file	The link to the picture of the a nimal.	String
album_update	The date of the previous updat e for the row.	Unknown. This column is em pty in all rows.
cDate	The date of the previous updat e for the row.	yyyy/mm/dd
shelter_address	As the name suggests.	String
shelter_tel	The telephone number of the s helter.	String

2. 寵物遺失啟事

Contents

This file contains the information of lost pets in Taiwan.

Source

行政院農業委員會資料開放平台 (coa.gov.tw)

Format

A json server which can be fetched with a HTTP request.

Rows

9999.

Columns

Name	Meaning	Format
chip_id	As the name suggests. This is the primary key of the table	A string of digits
name	The name of the pet.	String
type	The species of the pet.	{ 狗, 貓 }
sex	As the name suggests.	{ 公, 母 }
breed	As the name suggests.	String
color	As the name suggests.	A string of descriptions
appearance	As the name suggests.	A string of descriptions
feature	As the name suggests.	A string of descriptions
lost_time	As the name suggests.	yyyy/mm/dd
lost_place	As the name suggests.	String
owner_name	As the name suggests.	String
phone	The contact number of the owner	String
email	The contact email of the owner.	String
picture	A link to the picture of the pet.	String

3. 收容所資訊

Contents

This file contains the information of the public shelters in Taiwan.

Source

全國動物收容管理系統

Format

A table which can be obtained by web scraping.

Rows

32

Columns

Name	Meaning	Format
shelter_name	As the name suggests.	String
max_shelter	The maximum number of animals the shelter can contain.	Integer
num_shelter	The number of animals in the shelter.	Integer
light	The loading of the shelter. The heavier, the c loser to red,	{ red, yellow, gree n, blue }

4. 登記機構

Contents

This file contains the institutions which offer pet-related services or goods in Taiwan.

Source

寵物登記管理資訊網

Format

Tables in multiple pages which can be obtained by web scraping.

Rows

1972

Columns

Name	Meaning	Format
agency_id	As the name suggests. This is the primary key of the data.	Integer
agency_name	As the name suggests.	String
agency_address	As the name suggests.	String
contact_person	As the name suggests.	String
phone_number	As the name suggests.	String
email	As the name suggests.	String
distance	As the name suggests.	Unavailable. This is an advanced feature of the interface.
lat	The latitude of the institute.	Float
lng	The longitude of the institute.	Float

The columns lat and lng are not provided in the source. We convert the address into latitude and longitude with an external API. (find more details in Database - maintain our database 6.)

Normalization

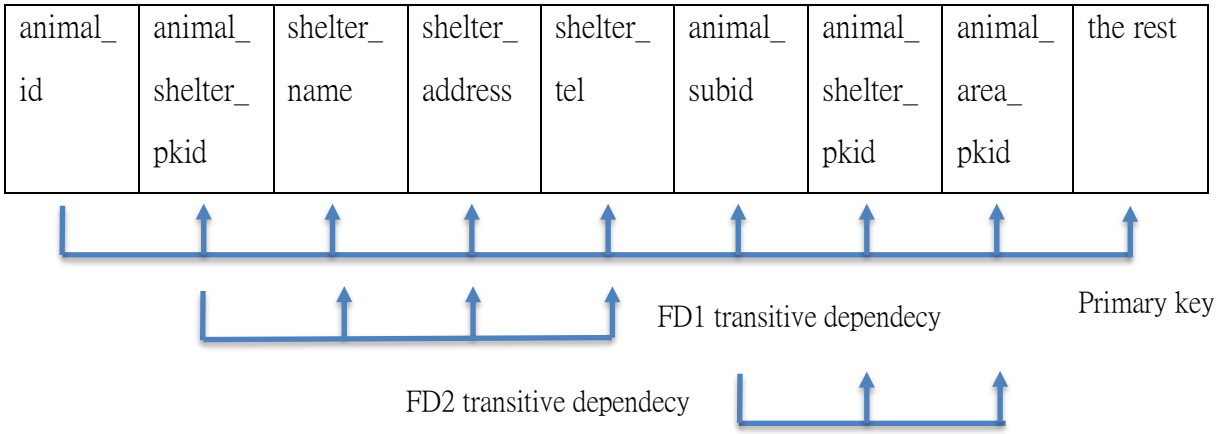
動物認領養:

1. Duplicate rows are deleted.
2. Columns which are blank in all rows are removed.
   {animal_title, animal_caption, animal_update, animal_place, cDate}
3. Decomposition

Dunctional dependencies:

FD1: {animal_shelter_pkid} → {shelter_name, shelter_address, shelter_tel}

FD2: {animal_subid} → {animal_shelter_pkid, animal_area_pkid}



The data is decomposed into 3 tables.

Table 1 **adopt_animal:**

All the columns in the data are loaded into this table. Except those on the right hand side of the 2 transitive dependencoes. Now **animal_shelter_pkid** becomes the foreign key to find shelters in **shelter** (more details below).

Table 2 **shelter:**

Attributes in FD1 and some columns in **收容所資訊** together create this table. Refer to **收容所資訊** below for more details.

Table 3 **animal_subid:**

Attributes in FD2 are loaded into the this table. However, this table is deleted in the end. The data here all indicates the location of the animals (more detailed but we do not need them), but the address of the shelters in shelter is good enough to tell that.

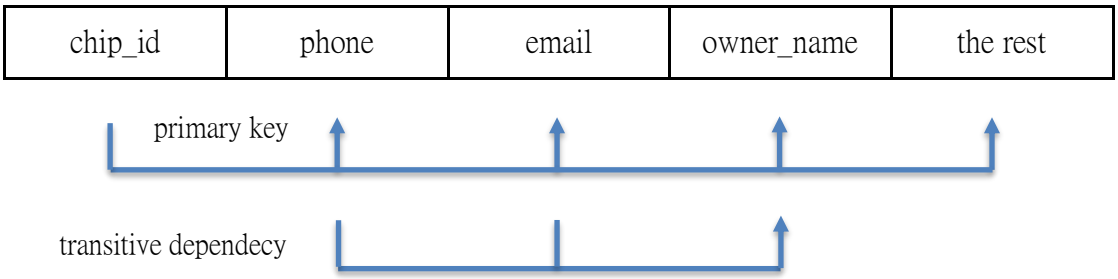
收容所資訊:

Both **收容所資訊** and **動物認領養** contain some columns related to shelters. The former has the capacity and loading of the shelter. Whereas the latter has the address and contact information. As a result, we combine them into one table **shelter**. On top of that, we also add 2 columns **can_help** and **need_help**, which will come in handy in the application. They are both boolean and are set to false initially. **can_help** indicates whether the shelter can help others, while **need_help** tells whether the shelter needs help.

寵物遺失啟事:

1. Duplicate rows are deleted.
2. Rows which have neither phone nor email are deleted. We think that such rows are redundant because people who find the lost pet will not be able to contact the owner anyway.
3. Functional dependency:

   { phone, email } -> { owner_name }



An owner may lose more than one pet, and there is a transitive dependency in the data. Hence, we decompose the data into 2 tables. Also, we define a new column **owner_id** to distinguish owners.

Table 1: lost_pet

We create this table with all the columns of the file except **owner_name, phone** and **email**. In addition, **owner_id** is added as the foreign key to find owners.

Table 2: owner

The table is composed of **owner_name, phone, email** and **owner_id** (primary key).

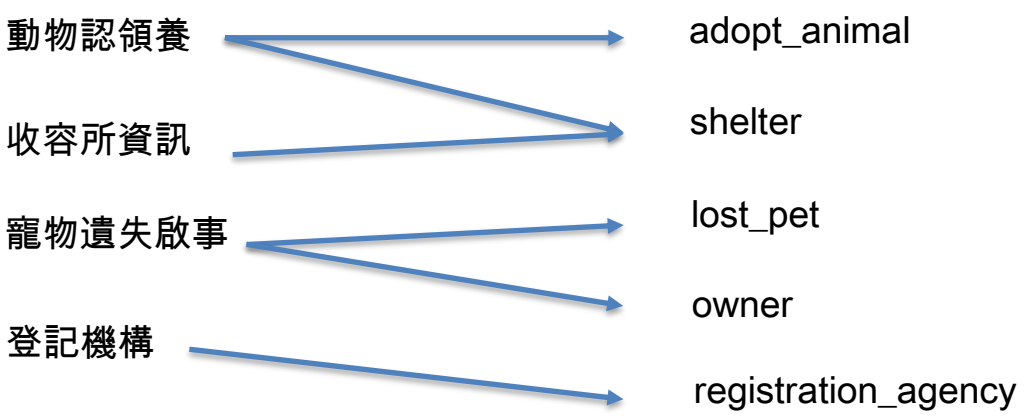
登記機構:

1. The column distance is removed because it is unavailable as mentioned in the previous section (Data).

2. After dropping the column, however, we hope to keep the feature (showing distance betwe en users and the institutes). To do so, we need to convert the address into latitude and long itude for every institute.

   In the beginning, we attempted to fetch them with API in the front end when loading the in terface. Nonetheless, it turned out to be too time consuming. In the end, we chose to downl oad and store them into the data. Therefore, we end up with 2 derived attributes in the tabl e - lat and lng.
3. No functional dependency at this point.
4. We created the table **registration_agency** with this file.

The diagram below summarizes how the data and tables are related.



Tables

We end up with the 5 following tables after normalization. The meanings are the same in **Data**. (for columns added in addition to the original data, please refer to the previous section for details)

```
mysql> describe adopt_animal;
+----------------------+--------------+------+-----+---------+----------------+
| Field                | Type         | Null | Key | Default | Extra          |
+----------------------+--------------+------+-----+---------+----------------+
| animal_id            | int          | NO   | PRI | NULL    | auto_increment |
| animal_shelter_pkid  | int          | NO   |     | NULL    |                |
| animal_kind          | varchar(3)   | YES  |     | NULL    |                |
| animal_sex           | varchar(1)   | YES  |     | NULL    |                |
| animal_bodytype      | varchar(10)  | YES  |     | NULL    |                |
| animal_colour        | varchar(8)   | YES  |     | NULL    |                |
| animal_age           | varchar(10)  | YES  |     | NULL    |                |
| animal_sterilization | char(1)      | YES  |     | NULL    |                |
| animal_bacterin      | char(1)      | YES  |     | NULL    |                |
| animal_foundplace    | varchar(100) | YES  |     | NULL    |                |
| animal_status        | varchar(10)  | YES  |     | NULL    |                |
| animal_remark        | varchar(300) | YES  |     | NULL    |                |
| animal_opendate      | varchar(14)  | YES  |     | NULL    |                |
| animal_closeddate    | date         | YES  |     | NULL    |                |
| animal_createtime    | varchar(14)  | YES  |     | NULL    |                |
| album_file           | varchar(100) | YES  |     | NULL    |                |
+----------------------+--------------+------+-----+---------+----------------+
```

```
mysql> describe lost_pet;
+------------+--------------+------+-----+---------+-------+
| Field      | Type         | Null | Key | Default | Extra |
+------------+--------------+------+-----+---------+-------+
| chip_id    | varchar(25)  | NO   | PRI | NULL    |       |
| name       | varchar(20)  | YES  |     | NULL    |       |
| type       | varchar(1)   | YES  |     | NULL    |       |
| sex        | varchar(1)   | YES  |     | NULL    |       |
| breed      | varchar(20)  | YES  |     | NULL    |       |
| color      | varchar(30)  | YES  |     | NULL    |       |
| appearance | varchar(20)  | YES  |     | NULL    |       |
| feature    | varchar(200) | YES  |     | NULL    |       |
| lost_time  | date         | YES  |     | NULL    |       |
| lost_place | varchar(50)  | YES  |     | NULL    |       |
| owner_id   | int          | NO   |     | NULL    |       |
| picture    | varchar(100) | YES  |     | NULL    |       |
+------------+--------------+------+-----+---------+-------+
```

```
mysql> describe owner;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| id    | int         | NO   | PRI | NULL    | auto_increment |
| name  | varchar(10) | YES  |     | NULL    |                |
| phone | varchar(40) | YES  |     | NULL    |                |
| email | varchar(50) | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
```

```
mysql> describe registration_agency;
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| agency_id      | int          | NO   | PRI | NULL    | auto_increment |
| agency_name    | varchar(50)  | YES  |     | NULL    |                |
| agency_address | varchar(100) | YES  |     | NULL    |                |
| contact_person | varchar(8)   | YES  |     | NULL    |                |
| phone_number   | varchar(30)  | YES  |     | NULL    |                |
| email          | varchar(100) | YES  |     | NULL    |                |
| lat            | decimal(8,5) | YES  |     | NULL    |                |
| lng            | decimal(8,5) | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
```

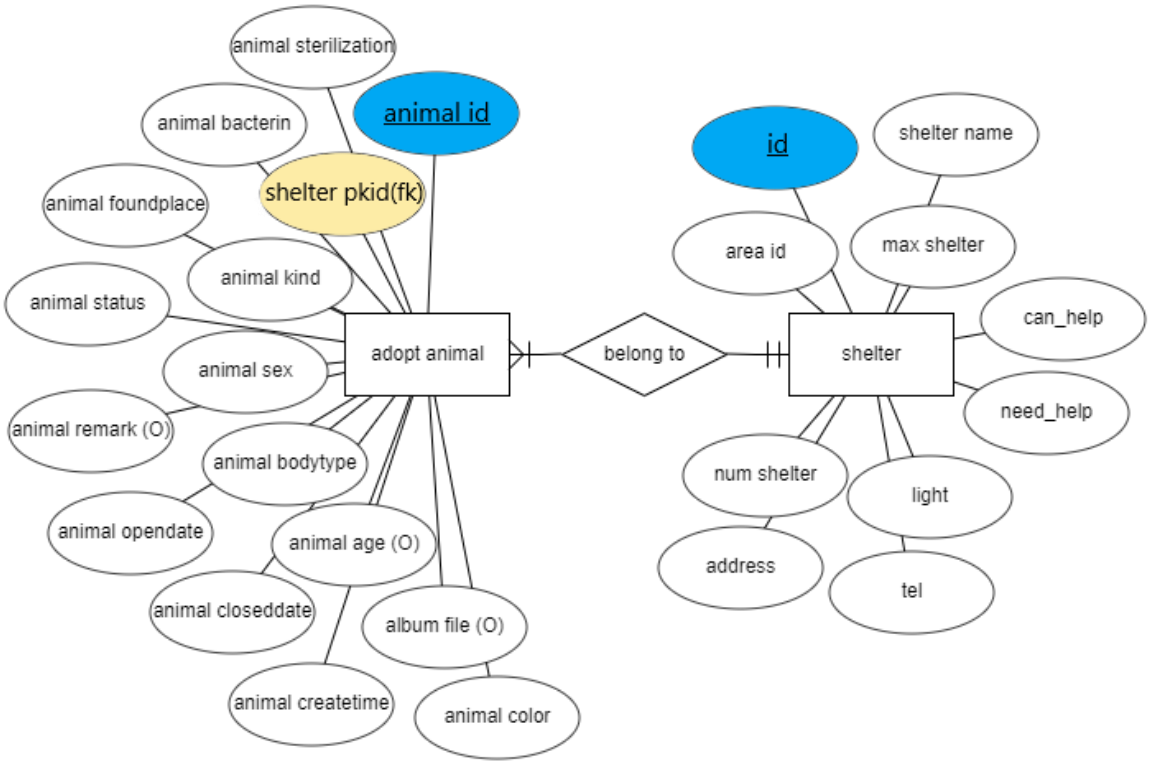
```
mysql> describe shelter;
+--------------+--------------+------+-----+---------+----------------+
| Field        | Type         | Null | Key | Default | Extra          |
+--------------+--------------+------+-----+---------+----------------+
| id           | int          | NO   | PRI | NULL    | auto_increment |
| shelter_name | varchar(50)  | YES  |     | NULL    |                |
| area_id      | int          | YES  |     | NULL    |                |
| max_shelter  | int          | YES  |     | NULL    |                |
| num_shelter  | int          | YES  |     | NULL    |                |
| light        | varchar(15)  | YES  |     | NULL    |                |
| address      | varchar(100) | YES  |     | NULL    |                |
| tel          | varchar(20)  | YES  |     | NULL    |                |
| can_help     | tinyint(1)   | YES  |     | 0       |                |
| need_help    | tinyint(1)   | YES  |     | 0       |                |
+--------------+--------------+------+-----+---------+----------------+
```

ER model

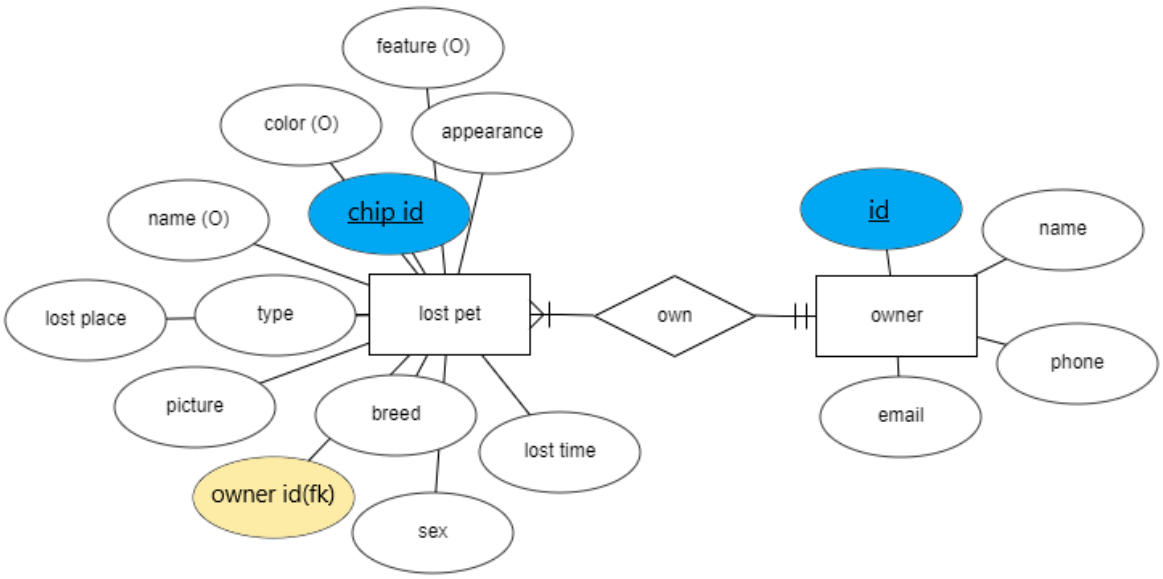
The underlined attributes in blue are primary keys.

Those in yellow are foreign keys which identify the other entity in the relation.

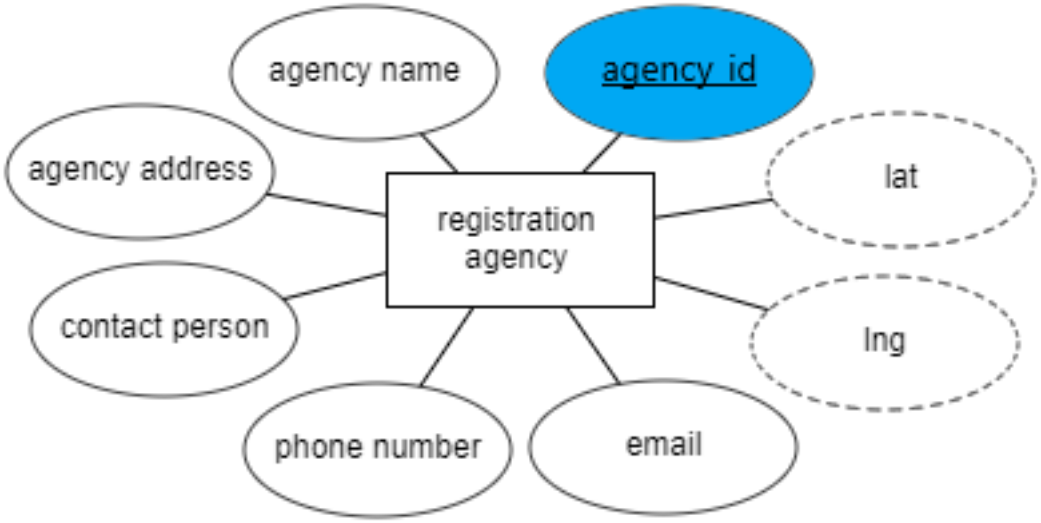
1. adopt animal & shelter



2. lost pet & owner



3. registration agency



Database

Maintain our database

1. Get the csv file which is used for our tables.
   We write a python to do web scraping to get the data from open sources.

```python
import requests;
import json;
import pandas as pd;
url = "https://data.coa.gov.tw/Service/OpenData/TransService.aspx?UnitId=QcbUEzN6E6DL";
response = requests.get(url);
df = pd.read_json(response.text);
df.to_csv("動物認領養.csv");
```

2. Delete the unnecessary blank in the data.
   We find that there will be some error when matching the data if there exist some prefix blanks in our data. we directly modify the csv file to delete the unnecessary blank in the data to prevent the problem.

3. Initialize our table.
   We first create four big tables corresponding to our four csv files. And then use the four big tables to conduct the normalization. We also change the empty value to "NULL" by the function of "NULLIF" in mysql to let the the decision of some statement correct.

```python
create table register_status_normalize(
    id int,
    county nvarchar(4),
    num_register_unit int,
    num_register int,
    num_remove int,
    num_transfer int,
    num_modify int,
    num_sterilization int,
    num_sterilization_remove int,
    num_free_from_sterilization int,
    primary key (id)
);

insert into register_status_normalize
select aa.id, aa.county, num_register_unit, num_register, num_remove, num_transfer,
       num_modify, num_sterilization, num_sterilization_remove,num_free_from_sterilization
from register_status rs,
     (select distinct substring(shelter_name,1,3) county, animal_area_pkid id from adopt_animal) aa
where rs.county = aa.county;
```

table normalization

```python
drop table if exists shelter_information;

create table shelter_information(
    id int,
    shelter_name varchar(50),
    max_shelter int,
    num_shelter int,
    light varchar(15),
    primary key (id)
);

load data local infile './收容所資訊.csv'
into table shelter_information
fields terminated by ','
/* enclosed by '"' */
lines terminated by '\n'
ignore 1 lines
(@a, @b, @c, @d, @e)
SET
id = NULLIF(@a,''),
shelter_name = NULLIF(@b,''),
max_shelter = NULLIF(@c,''),
num_shelter = NULLIF(@d,''),
light = NULLIF(@e,'');
```

unnormalized table

We finish the process of normalization by sourcing each table in sequence.

After the normalization completing, we just save the tables splitted producing during the process of normalization and drop the original four big tables.

```
source adopt_animal.sql;
source adopt_animal_normalize.sql;
source animal_subid_normalize.sql;
source lost_pet.sql;
source owner.sql;
source lost_pet_normalize.sql;
source registration_agency.sql;
source registration_agency_normalize.sql;
source shelter_information.sql;
source shelter_normalize.sql;
select 'normalize complete';
drop table adopt_animal;
drop table lost_pet;
drop table registration_agency;
drop table shelter_information;
select 'drop table complete';
alter table adopt_animal_normalize rename to adopt_animal;
alter table lost_pet_normalize rename to lost_pet;
alter table registration_agency_normalize rename to registration_agency;
select 'rename table complete';
```

4. Select a proper attribute to be the primary key for our tables.
   Most of our tables have columns called ….._id. The value of column is auto increment. Those columns are relevant between most of our tables. As a result we choose this type of column to do the process of normalization.
   We choose that type of column to be the primary keys or foreign keys of the table which is splitted after normalization, because those column will be still auto increment. This is a good attribute in terms of being a primary key.
   By doing so, this makes it easy to manage our database.

5. When the user wants to post a post to ask other people to help find their lost pet.
   the method we take: In order to prevent users posting duplicate posts of their lost pet. We ask the user to register our website first and then they are allowed to post. All the information is filled on our website. After the users submit their post the website will call our sql_server to connect to our database to do the query of insertion to the table lost_pet.

6. Calculate the latitude and longitude of the address of register_agency first.
   In the original csv file the table does not have the columns for the latitude and longitude for the address in the table and if we calculate that every time it will waste so much time. As a result, we decide to preprocess the data first insert two extra column in the csv file, called lat and lng to store latitude and longitude
   The whole steps are that we use import "pandas" to read the csv file. And then get

```
address_list = file["地址"];
position = [];
for address in address_list:
    if (address == "null room"):
        position.append({"address": address, "lat": 0, "lng": 0});
        continue;

    response = requests.get(url + address + "&apiKey=" + key);
    data = response.json();

    if (len(data["items"]) == 0):
        position.append({"address": address, "lat": 0, "lng": 0});
        continue;

    p = data["items"][0]["position"];
    p["address"] = address;
    print(p);
    position.append(p);
lat = [];
lng = [];
for pos in position:
    if (pos["lat"] == 0): lat.append(None);
    else: lat.append(pos["lat"]);

    if (pos["lng"] == 0): lng.append(None);
    else: lng.append(pos["lng"]);
file["lat"] = lat;
file["lng"] = lng;
file = file.drop(columns=["Unnamed: 0", "Unnamed: 0.1"]);
file.head()
file.to_csv("登記機構.csv");
```

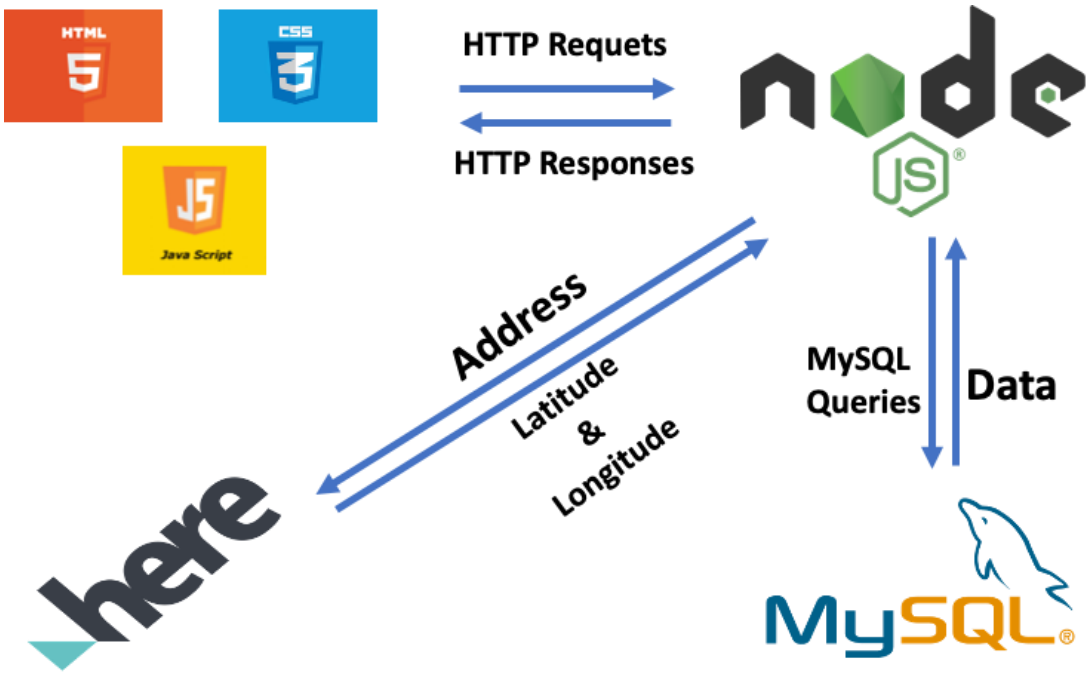
the data in the column of "address". Concatenate the string to meet the request of the HER E API. Later, we can get the latitude and longitude for the provided address. Finally, add t wo more columns to the original csv file, called lat and lng to store the data in the same se quence that we read the file.

7. Avoid duplicate owner in table owner

   Table "owner" was created because we found that some of the owners may have lost m ore than one pet or have no information. However, unexpected difficulties arose while we conducted the queries. In the beginning, we expected that using the instruction "insert ign ore into" to avoid duplicate rows in owner will work, but it turned out to be that "ignor e" can't detect duplicate rows with one of whose attributes is empty(null). So, here com es the complicated query below, for the purpose of filtering those duplicate rows I mention ed before. The idea is to first list all duplicate rows, including those with null attributes, th en we group them by name, phone, and email, making sure each group represents a distinc t owner. Finally, select rows with minimum id from each group, use "not in" to remove them, and delete the rest rows the query found. This procedure can avoid the problem we met before.

```
delete from owner
where id in
(select id
from    (select distinct o1.id, o1.name, o1.phone, o1.email
    from owner o1,owner o2
    where o1.email = o2.email and o1.phone=o2.phone and o1.name=o2.name and o1.id!=o2.id
    union
    select distinct o1.id, o1.name, o1.phone, o1.email
    from owner o1,owner o2
    where o1.email is null and o2.email is null and o1.phone=o2.phone and o1.name=o2.name and o1.id!=o2.id
    union
    select distinct o1.id, o1.name, o1.phone, o1.email
    from owner o1,owner o2
    where o1.email = o2.email and o1.phone is null and o2.phone is null and o1.name=o2.name and o1.id!=o2.id) as temp
where id not in
(select min(id) as id
from (select distinct o1.id, o1.name, o1.phone, o1.email
    from owner o1,owner o2
    where o1.email = o2.email and o1.phone=o2.phone and o1.name=o2.name and o1.id!=o2.id
    union
    select distinct o1.id, o1.name, o1.phone, o1.email
    from owner o1,owner o2
    where o1.email is null and o2.email is null and o1.phone=o2.phone and o1.name=o2.name and o1.id!=o2.id
    union
    select distinct o1.id, o1.name, o1.phone, o1.email
    from owner o1,owner o2
    where o1.email = o2.email and o1.phone is null and o2.phone is null and o1.name=o2.name and o1.id!=o2.id) as temp
group by name,phone,email));
```

Connection



The graph above briefly shows how the application is connected to the database. Please refer to th e following for further details.

Front end:

The 3 basic web design languages (HTML, CSS, Javascript) are implemented to provide a basic outlook and enable interactions with users. When a user attempts to do a CRUD, the front end sends a http request to the server (back end), which will then return the requested data if there is any. Also, most of the exceptions of user inputs are handled here.

Back end:

There are 2 servers created by node.js.

1. A server (sql_server.js) is created with node.js to establish the connections betwee n the front end and the database. It runs at port 3000, waiting for requests from the front end. Requests from different interfaces go to their corresponding route for fur ther processing.

   Upon receiving a request, the server generates a corresponding SQL query based o n the information given by the request body (elaboration will be provided later on). The query will then be sent to the database and the data will be returned to the serv er (if there is any).

2. Another server (pos_server.js) is used to communicate with the API. It runs at port 8000, waiting for requests from the front end. An address provided by the user is a ll the request body contains.

   Error status code 999 is returned if the address is invalid or the API is unable to co nvert it.

Database:

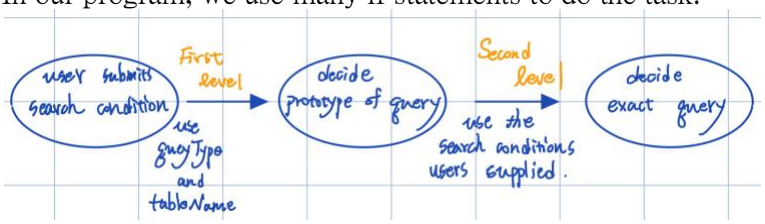
We choose MySQL as the database.

API:

The api we used is called "HERE Geocoding and Search api". It is used to find the geo-coordinates of a known address, place, locality or administrative area. It supp orts multiple languages. The request url will be like " https://geocode.search.hereapi.com/v1/ge ocode?q={address to be geocoded} &apiKey={YOUR_API_KEY}". The return data type is json file. The data for latitude and longitude i n the json file are stored in the tag of "address".

Backend process queries from application:

We let the users choose to fill out their search condition. In order to prevent unexpected in put, we let the user choose from the list, but not let them type whatever they want. We classify que ries into different types. We call those types in our nodejs programs as "queryType". Different t ables support different types of queries. We use two levels to decide what exact query should be sent to our database from our application. In the fi rst level, we check "queryType" and "tableName" to decide which prototype of query should be executed. However, this is not enough to decide the exact query, bec ause users do not need to fill out all search conditions. So we use the second level to solve the problem. The first level decides the prototype of our query. Condition s will be appended to it if the user supply the corresponded search condition. In our program, we use many if statements to do the task.



the flow of deciding a query

The way we decide the exact query. Our query needs the search conditions that the user provides. Our strategy is we append the condition on the where clause of the prototype query if the search c ondition is provided. For example, the structure will look like the photo. if the “queryType” is “FILTER” and “tableName” is “shelter”, then the prototype of query will be “select * from shelter”. Keep going on, the inner if statement decide what condition should be present in the where clause. We use string concatenation to concatenate conditions to the where clause.

the way we decide the exact query →

```
if (reqBody.queryType == "FILTER" && reqBody.tableName == "shelter") {
  query = `SELECT *\nFROM ${reqBody.tableName}`;
  var where = false;

  if (reqBody.address) {
        const addressPattern = `%${reqBody.address}%`;
        query = `${query}\nWHERE address LIKE '${addressPattern}'`;
        where = true;
  }

  if (reqBody.name) {
        const namePattern = `%${reqBody.name}%`;
        query = `${query}${where ? " AND\n" : "\nWHERE"} shelter_name LIKE '${namePattern}'`;
        where = true;
  }

  if (reqBody.region != "無") {
        const regionPattern = `${reqBody.region}%`;
        query = `${query}${where ? " AND\n" : "\nWHERE"} address LIKE '${regionPattern}'`;
        where = true;
  }

  if (reqBody.needHelp != "無") {
        const needHelp = (reqBody.needHelp == "是") ? 1 : 0;
        query = `${query}${where ? " AND\n" : "\nWHERE"} need_help = ${needHelp}`;
        where = true;
  }

  if (reqBody.canHelp != "無") {
        const canHelp = (reqBody.canHelp == "是") ? 1 : 0;
        query = `${query}${where ? " AND\n" : "\nWHERE"} can_help = ${canHelp}`;
        where = true;
  }

  if (reqBody.sort)
        query = `${query}\nORDER BY num_shelter/max_shelter ASC;`
```

Application

Run the App

0. Prerequisites:
   - Download node.js at https://nodejs.org/en/download/

1. Browser settings:
   - Disable cross-origin restrictions
     (some browsers forbid client and server from running on the same device)

2. Commands:

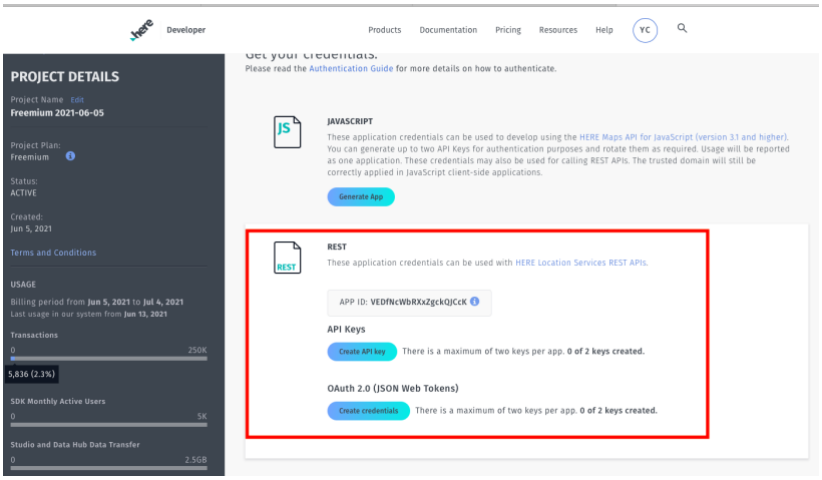
   Commands below are run in the app folder, except ($ source database.sql)
   (corresponding to the folder on the Github repo)

   If you are starting the app for the first time
   - run the command below to install the node modules
     $ npm install
   - run the following command in MySQL to build the database
     $ source database.sql
   - Set the password and name of your database in sql_server.js
     (line 12 & 13)

   To start the server, run the following command
   $ node sql_server.js

   If you are using "根據距離排序" in "登記機構"

   Please sign up and create a new key for REST
   (the one in the red box)
   https://developer.here.com/projects/PROD-4010ac1d-597c-44c6-898b-96d794ed43c5



In pos_server.js, replace the key with the new one at line 9
(the keys typically expire in a couple of hours)

Run the command below in a new terminal tab

```
$ node pos_server.js
```

3. Open the websites:

open { interfaces name }.html in your browser

Interfaces

Following are some points we want you to know before going on.

(1) There are 6 interfaces in total, each serving for different purposes and users.

(2) For all the filters which offer select options, “無” is selected by default. If it is s elected, the contents will not be filtered by that category.

(3) There is a separate file (exception.js) in the app folder which keeps all the functions neede d for exception handling on the front end. For all the exception handling cases, there will b e a remark specifying which function it uses next to it. Please refer to the file (exception.j s) for further details.

(4) All the HTML input boxes have a max length limit (if needed) so that user inputs do not e xceed the corresponding data in the database.

(5) The following applies to all of the edit buttons.

if the button is clicked when the text inside is “edit”:

- The text becomes “done”.
- The input boxes appear.
- Users can start editing.

if the button is clicked when the text inside is “done”:

- The text becomes “edit”.
- The input boxes disappear.
- The changes are submitted.

(6) For each interface, we do not fetch all the data which meets the requirements because the n umber of records are not small in some tables. Images will take too long to load.

1. Home

Contents

This page displays animals (from adopt_animal) kept by the public shelters in Taiwan.

Users

Those who are looking for animals to adopt.

Functions

Users can make use of the filter to search for the animals. Following are the inputs for the filter.

- 動物種類

  Users can select 1 of the following: “無”, “狗”, “貓”, “其他”.

  - Match Target: animal kind
  - Match Criteria: exact match (“其他” looks for animals which are n either dog nor cat)

- 體型

  Users can select 1 of the following: “無”, “SMALL”, “MEDIUM”, “BIG”.

  - Match Target: animal body type
  - Match Criteria: exact match

- 性別

  Users can select 1 of the following: “無”, “M” (for male), “F” (for female)

  - Match Target: animal sex
  - Match Criteria: exact match

- 顏色

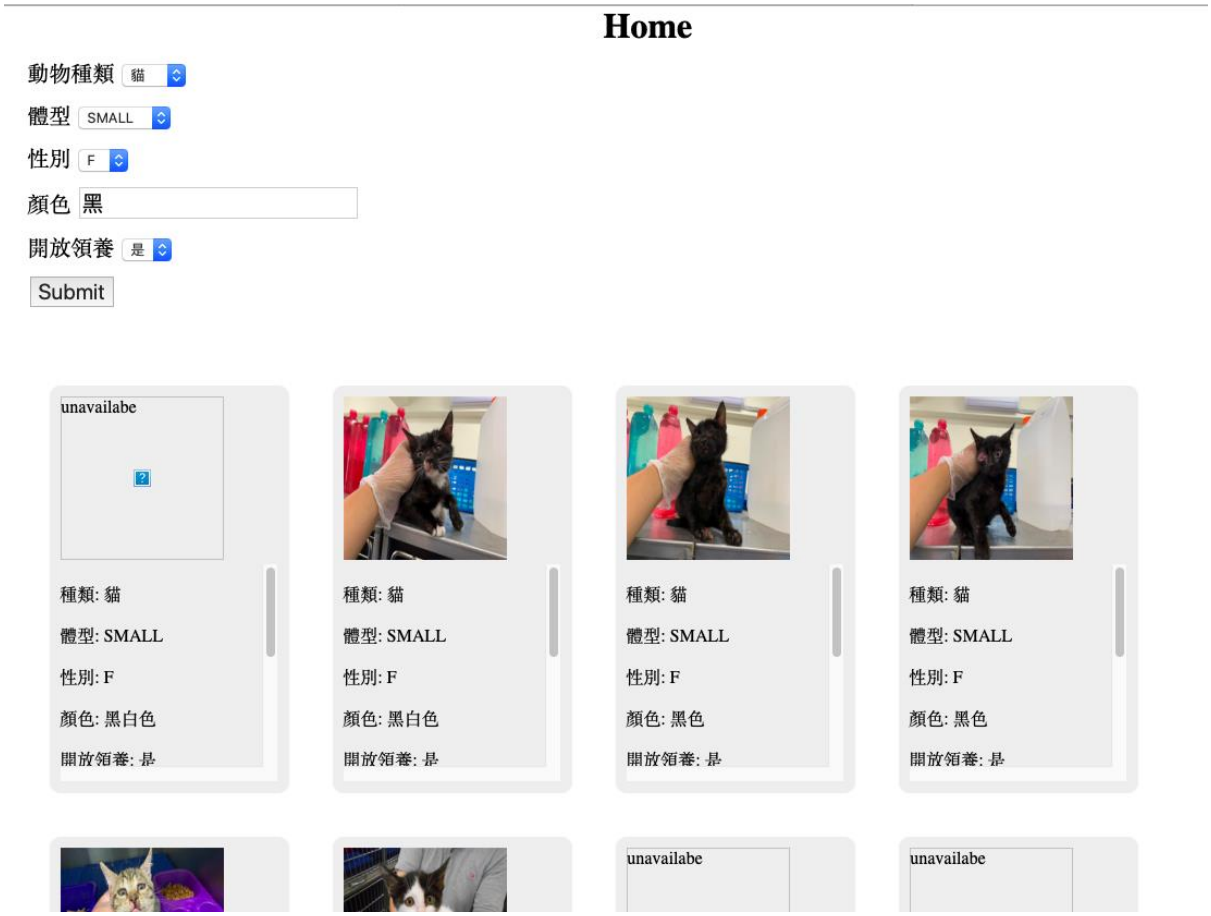
  Enter the color of the animal.

  - Match Target: animal color
  - Match Criteria: regular expression (“%input%”)

- 開放領養

  Users can select 1 of the following: “無”, “是”, “否”.

- Match Target: animal status
- Match Criteria: “OPEN” for “是”, the rest for “否”



Sample SQL

```
SELECT *
FROM adopt_animal AS A, shelter AS S
WHERE A.animal_shelter_pkid = S.id AND
 A.animal_kind = '貓' AND
 A.animal_bodytype = 'SMALL' AND
 A.animal_sex = 'F' AND
 A.animal_colour LIKE '%黑%' AND
 A.animal_status = 'OPEN'
ORDER BY animal_id DESC
LIMIT 50;
```

2. 全台公立收容所

Contents

This page displays all the public shelters in Taiwan.

Users

The staff of the public shelters.

Motivations

It is common for the number of stray animals to be overwhelming for public shelters. Sometimes, they have no choice but to execute euthusansia or sacrifice the living quality in order to take on newcomers.

However, when taking a closer look at the data, we find that some of them actually do have adequate idle capacity. Therefore, 2 extra columns are created in addition to the original data: “需要支援” and “可提供支援”. They should make it easier to look for helpers or those who need help. We expect that the shelters make use of them to reallocate the animals they keep. Hopefully, the above-mentioned issue can be solved by reducing idle resources.

Functions

Users can filter and sort the contents with the filter.

- **地址**

  Enter a part of (or complete) an address.
  - Match Target: shelter address
  - Match Criteria: regular expressions (“%input%”)

- **名稱**

  Enter a part of (or complete) the name of the shelter.
  - Match Target: shelter name
  - Match Criteria: regular expressions (“%input%”)

- **地區**

  Select 1 of the provinces in Taiwan.
  - Match Target: shelter address

- Match Criteria: regular expressions (“province%”)

- 需要支援

  Select 1 of the following: “無”, “是”, “否”.

  - Match Target: whether the shelter needs help from others
  - Match Criteria: exact match

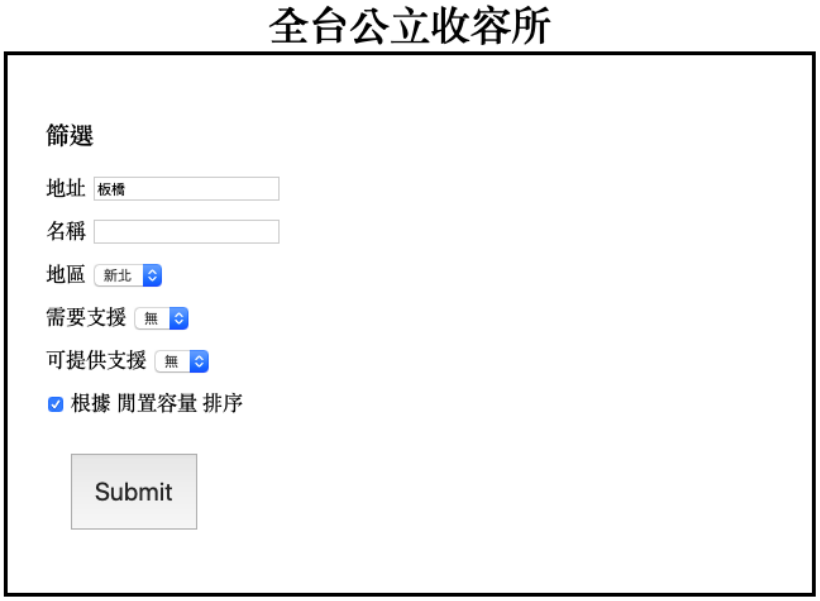
- 可提供支援

  Select 1 of the following: “無”, “是”, “否”.

  - Match Target: whether the shelter is willing to help others
  - Match Criteria: exact match

- 根據閒置容量排序

  Click the check box to sort the shelters by their idle capacity (“容量” - ” 已收容數量”) in descending order so that users can find those who are most likely to help or need help easily.



收容所	地址	電話	容量	已收容數量	需要支援	可提供支援
新北市板橋區公立動物之家	新北市板橋區板城路28-1號	02-89662158	347	195	否	否
新北市政府動物保護防疫處	新北市板橋區四川路一段157巷2號	02-29596353	20	33	否	否

Sample SQL

```
SELECT *
FROM shelter
WHERE address LIKE '%板橋%' AND
 address LIKE '新北%'
ORDER BY max_shelter - num_shelter DESC;
```

3. 收容所管理

Contents

This page displays the details of a public shelter. The staff can find their shelter by logging in with the unique id of it. After log in, they can view and edit the information of the shelter, including the animals it keeps.

Users

The staff of the public shelters.

Motivations

All the animals are put in one csv file (動物認領養.csv) in the raw data, regardless of which shelters they are staying. We believe that another interface is needed for the staff to manage the shelter more efficiently.

Functions

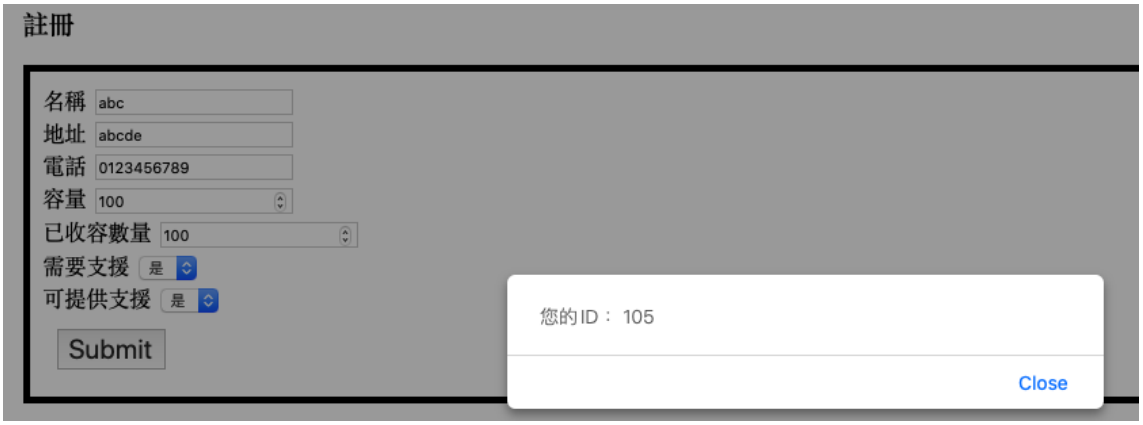
a. register

A new shelter must be registered before it can enjoy the features of this interface. After filling in the information, click the submit button. A new id will be created automatically by the database (auto_increment) and sent back to the front end.

```
db.query(getQuery(req.body), (err, rows) => {
    if(err) throw err;

    if (req.body.queryType == "INSERT") {
        db.query("SELECT MAX(id) AS id FROM shelter;", (err, id) => {
            if(err) throw err;
            res.send(id);
        })
    }
```

A window will pop up to alert the new id.



Sample SQL

```
INSERT INTO shelter (shelter_name, area_id, max_shelter, num_shelter, light, address, tel, need_help, can_help)
VALUES ('abc', 0, 100, 100, '', 'abcde', '0123456', 1, 1);
```

The restrictions are:

- “名稱”, “地址”, “電話” are required. (no specific function is used, simply check if it is undefined)
- “電話” contains only digits, whitespace or “-”. (handled by validPhoneNumber)
- “容量”, “已收容數量” must be a non-zero integer. (handled by validIntRange)

b. log in

Staff of a shelter can log in with the unique id received upon registration. If the id does not exist (the SELECT query returns empty), the backend server returns an error status code. As a result, a window with an error message will pop up when the submit button is clicked. Otherwise, the information of the shelter and all the animals it keeps will be displayed.

c. edit / delete shelter information

To update the information, click the edit button. Click again to submit the updates. The restrictions are the same as in registration.

The users can click the delete button to delete the shelter. However, this will also delete every animal kept in the shelter (from adopt_animal) automatically. By doing so, we can ensure the integrity of data. After all, animals should not exist in the database if they belong to no shelter.

Sample Update Shelter SQL

```
UPDATE shelter
SET shelter_name = '新北市淡水區公立動物之家',  address = '新北市淡水區下圭柔山91之3號',  tel = '02-26267558',  max_shelter = '85',  num_shelter = '56',  need_help = 1,  can_help = 1
WHERE id = 55;
```

Sample Delete Shelter SQL

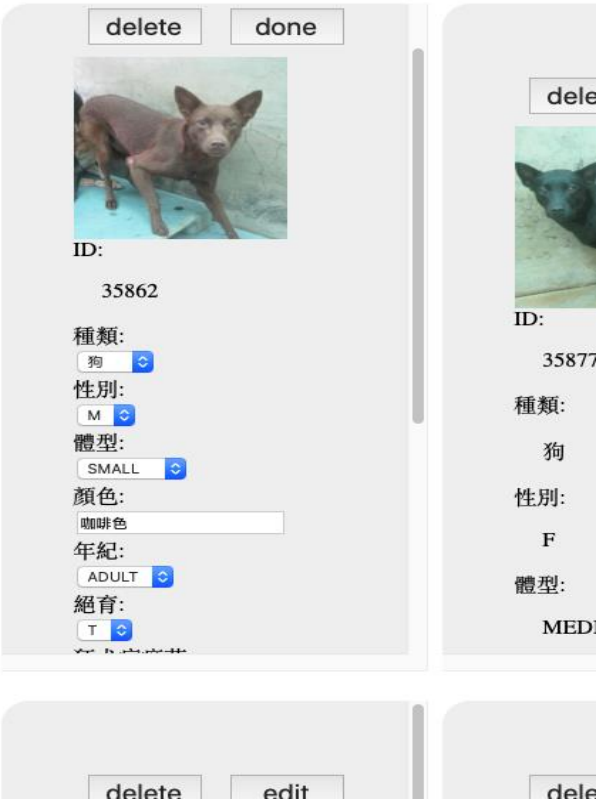
```
DELETE FROM adopt_animal
WHERE animal_shelter_pkid = 105;

DELETE FROM shelter
WHERE id = 105;
```

e. edit / delete animal

To update the animal information, click the edit button. Click again to submit the changes. The restrictions on the inputs are the same as in adding a new one. However, everything can be edited except id because it is the primary key of the table.

To delete an animal, click the delete button. It will not be displayed in “收容動物” and the Home interface anymore after refresh.



Sample Delete Animal SQL

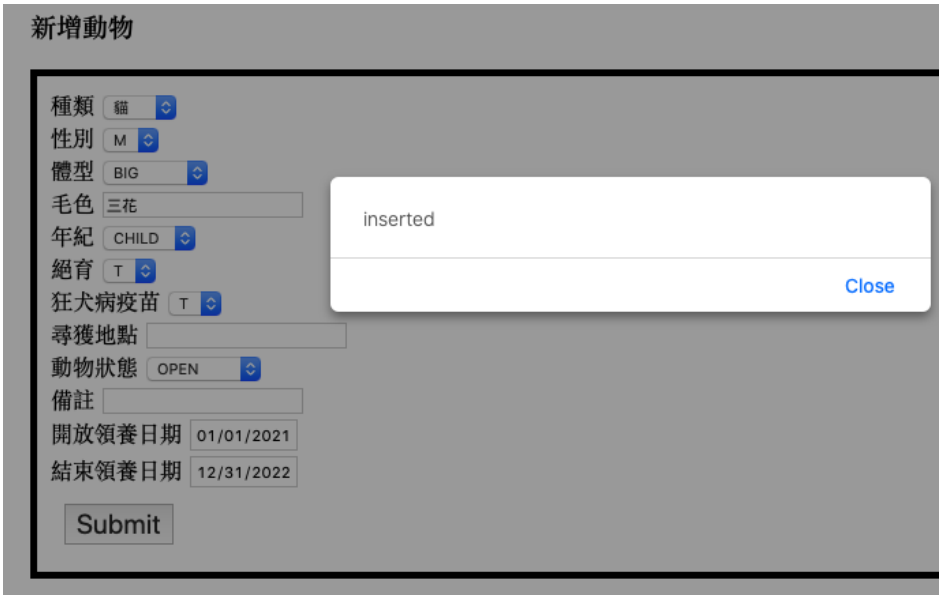
```
DELETE FROM adopt_animal
WHERE animal_id = 205825;
```

Sample Edit Animal SQL

```
UPDATE adopt_animal
SET animal_kind = '狗',  animal_sex = 'M',  animal_bodytype = 'SMALL',  animal_colour = '三花',  animal_age = 'ADULT',
 animal_sterilization = 'T',  animal_bacterin = 'T',  animal_foundplace = 'undefined',  animal_status = 'OPEN',  animal
_remark = '',  animal_opendate = '2021-01-01',  animal_closeddate = '2022-12-30'
WHERE animal_id = 205824;
```

f. add a new animal to the shelter

To add a new animal, scroll down to the very button of the page. Fill in the information about the animal and click submit, it will then appear in “收容動物” as well as the Home interface. An id for the animal will be created automatically. The only restriction is that “開放領養時間” must not be later than “結束領養時間”. (handled by compareDates)



Sample SQL

```
INSERT INTO adopt_animal (animal_shelter_pkid, animal_kind, animal_sex, animal_bodytype, animal_colour, animal_age, ani
mal_sterilization, animal_bacterin, animal_foundplace, animal_status, animal_remark, animal_opendate, animal_closeddate
)
VALUES (107, '貓', 'M', 'BIG', '三花', 'CHILD', 'T', 'T', '', 'OPEN', '', '2021-01-01', '2022-12-31');
```

4. 寵物協尋

Contents

This page displays the lost pets and their information, including that of the owners.
p.s. Most of the links to the image of the pets are invalid so a lot of the images displayed in the interface shows "unavailable".

Users

- Owners whose pets are lost
- People who find the lost pets trying to contact the owners

Motivations

Pets getting lost is quite common. Worried owners might want to hand out fliers or spread the information on social media. However, a designated platform might be a more effective solution.

Functions

a. Display the lost pets

```
SELECT *
FROM lost_pet AS P, owner AS O
WHERE P.owner_id = O.id
LIMIT 50;
```

b. Filter the lost pets

Users can look for the lost pets with the help of the filter.

- **晶片號碼**

  Enter a unique chip id of the pet.
  - Match Target: pets chip id
  - Match Criteria: exact match

  p.s. If this box is not blank, the database will search by chip id only. After all, no 2 pets share the same id.

- **種類**

  Select 1 of the following: 貓, 狗, 其他
  - Match Target: pets type
  - Match Criteria: exact match

- **品種**

  Enter the breed of the pet.

  - Match Target: pets breed
  - Match Criteria: regular expressions (“%input%”)

- **性別**

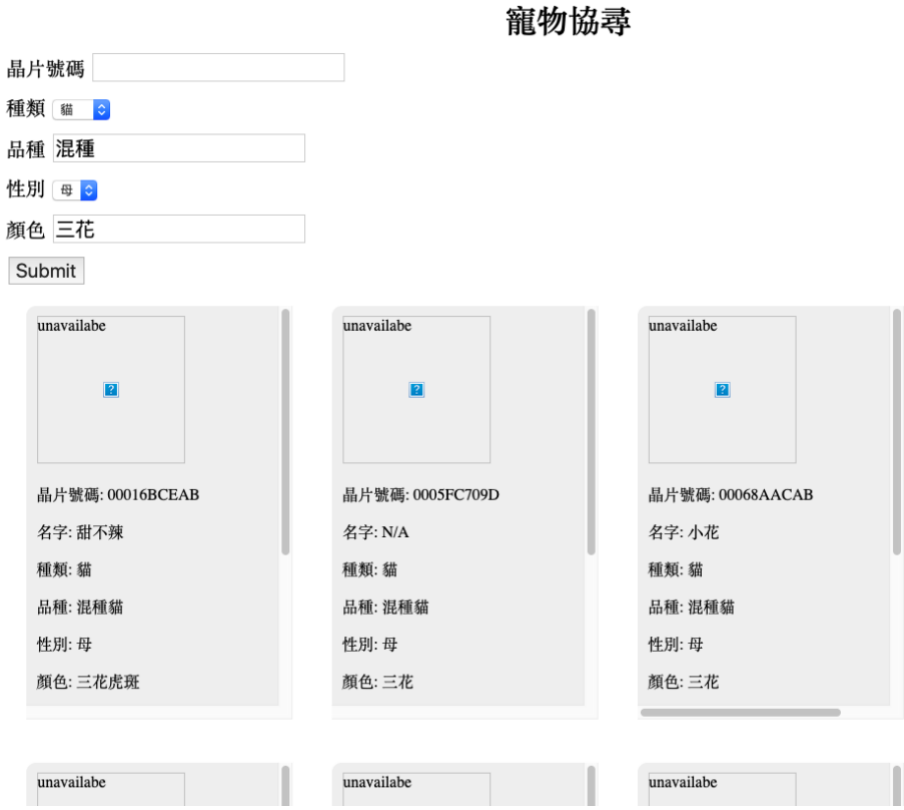
  Select 1 of the following: 公, 母

  - Match Target: pets sex
  - Match Criteria: exact match

- **顏色**

  Enter the color of the pet.

  - Match Target: pets color
  - Match Criteria: regular expressions (“%input%”)



Sample SQL

```
SELECT *
FROM lost_pet
WHERE type = '貓' AND
 breed LIKE '%混種%' AND
 sex = '母' AND
 color LIKE '%三花%'
```

5. 飼主

Contents

The information of the owner and his / her lost pets are displayed here. Owners can log in to this page and edit their contact information (name, phone number, email). In addition, owners can post / edit their lost pets here, which will then appear on the 寵物協尋 page.

Users

Owners whose pets are lost.

Motivations

In the raw data (寵物遺失啟事.csv), there are many duplicate lost pets. Anxious owners probably forgot that they have posted the info and do it repeatedly. We believe that this is an issue which we must take care of. A designated interface for the owners should help them manage the information more easily.

Functions

a. register

A new owner must register before he / she can enjoy the features of this interface. After filling in the information, click the submit button. A window will pop up and alert the new id.

- "稱謂" is required. The user must leave at least 1 contact information, "聯絡電話" or "電話". (no specific function is used, simply check if it is undefined)
- "電話" contains only digits, whitespace or "-". (handled by validPhoneNumber)

Sample SQL

```
INSERT INTO owner (name, phone, email)
VALUES ('abc', '123456789', 'abc@gmail.com');
```

b. log in

Owners can log in with the unique id received upon registration. If the id does not exist (the SELECT query returns empty), the backend server returns an error status code. As a result, a window with an error message will pop up when the submit bu tton is clicked. Otherwise, the information of the owner and all the pets posted will be displayed.

Sample SQL

```
SELECT *
FROM owner
WHERE id = 1233;


SELECT *
FROM lost_pet
WHERE owner_id = 1233;
```

c. edit / delete owner information

To update the information, click the edit button. Click again to submit the updates. The restrictions on the inputs are the same as in registration.

Owners can click the delete button to delete the account. However, this will also delete every lost pet which belongs to him or her. (from lost_pet) automatically. By doing so, we can ensure the integrity of data. After all, an animal without an owner cannot be lost or even a pet.

Edit owner information

```
UPDATE owner
SET name = '周 先生/小姐',  phone = '0922836110',  email = ''
WHERE id = 1233;
```

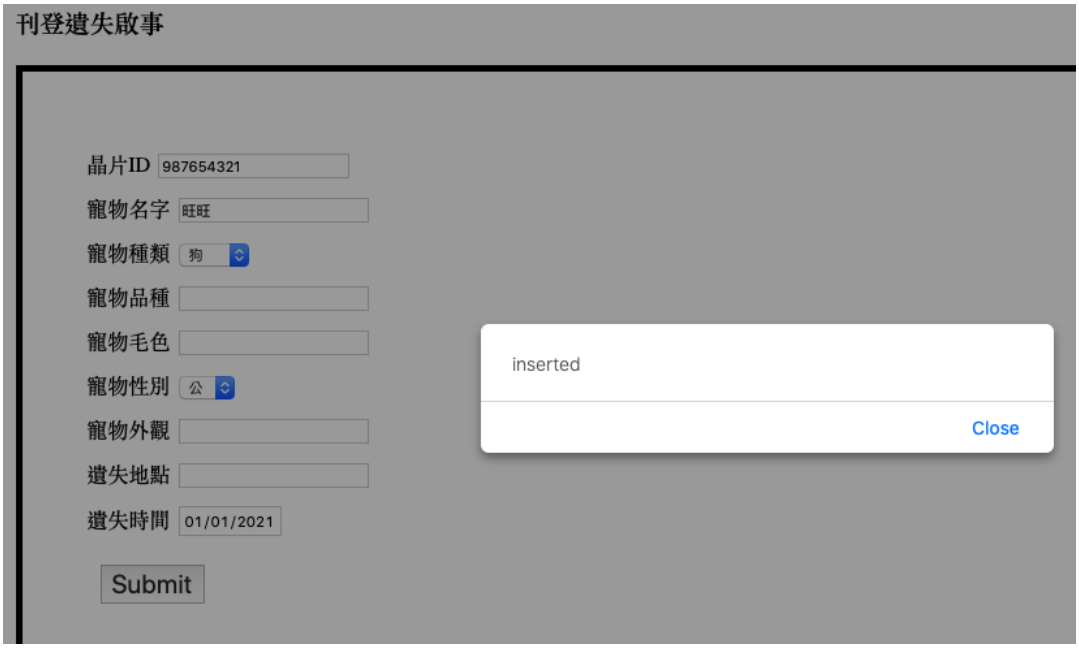
Delete account

```
DELETE FROM lost_pet
WHERE owner_id = 1233;

DELETE FROM owner
WHERE id = 1233;
```

d. add a new lost pet

To add a new lost pet, scroll down to the very button of the page. Fill in the information and click submit, it will then appear in ”您的遺失寵物” as well as in the “寵物協尋” interface.



Sample SQL

```
INSERT INTO lost_pet (chip_id, name, type, sex, breed, color, appearance, lost_time, lost_place, owner_id, picture)
VALUES ('987654321', '旺旺', '狗', '公', '', '', '', '2021-01-01', '', '1234', '');
```

The only restriction is that “晶片ID” must not exist in the table prior to the insertion. If it does, the server will send an error status code (999). In the interface, a window with an error message will pop up to prompt the user to check the id again.

Below is how the server handles duplicate chip id. It checks if the given chip id exists (whether the length of the returned data is 0) before insertion.

```
app.post("/lost/post", (req, res) => {
    if (req.body.queryType == "INSERT") {
        db.query(`SELECT * FROM ${req.body.tableName} WHERE chip_id = ${req.body.id}`, (err, rows) => {
            if (rows.length != 0)
                res.sendStatus(999);

            else {
                db.query(getQuery(req.body), (e, r) => {
                    if(e) throw e;
                    res.send(r);
                });
            }
        })
    }
```

e. edit / delete a lost pet

To update the animal information, click the edit button. Click again to submit the changes. However, “晶片ID” cannot be edited in order to keep the integrity of the database (this is the primary key of the table, we do not want users to update it). Owners must delete and add it back if they want to change “晶片ID” .

To delete a lost pet, click the delete button. It will not be displayed here and in the “寵物協尋” interface anymore after refresh.

6. **相關機構**

Contents

This page displays institutes which offer pet-related services or goods.

Users

- Pet owners
- Insitute owners

Motivations

Taking care of little lives is way more difficult than just feeding them. Pets can have all ki nds of unexpected needs. This interface helps users to explore nearby institutes which offer pet-related services or goods.

Functions

a. filter & sort

Users can search for the institutes by utilizing the filter.

- **地址**

  Enter a part of (or complete) an address.

  - Match Target: institutes address
  - Match Criteria: regular expressions (“%input%”)

- **名稱**

  Most of the institutes in the table indicate what kind of services / goods they offer in the names. To look for a specific type of instit ute, pet salon for instance, users can try entering “美容”.

  - Match Target: institutes name
  - Match Criteria: regular expression (“%input%”)

- **地區**

  Users can select a province (in Taiwan) to limit the location of the displaye d contents.

  - Match Target: institutes address
  - Match Criteria: regular expression (“input%”)

- **根據距離排序**

Filtering by “地區” probably is not effective enough. Users can e xplore the neighborhood by checking the box and entering their address (the more complete, the better).

If the box is checked but the address is invalid (blank or unidentifiable by t he API), a window with an error message will pop up to prompt the users t o check the address again. Otherwise, the contents will be filtered and sort ed by their distance to the given address ascendingly. And the rightmost co lumn will show the distance in kilometers.

p.s. Remember to make sure that the key for the API is not expired (as inst ructed in Run the App) if you want to use this feature.

Sample SQL

```
SELECT agency_id, agency_name, agency_address, contact_person, phone_number, email, 2 * PI() * 6371.3 / 360 * SQRT(POW(
(24.78929 - lat), 2) + POW((121.00003 - lng), 2)) AS distance
FROM registration_agency
WHERE lat IS NOT NULL AND lng is NOT NULL AND
 agency_name LIKE '%美容%'
ORDER BY distance ASC
LIMIT 50;
```

b. add a new institute

An owner of an institute can create a record in “新增機構”. The only res triction is that none of the input boxes can be blank.


```
INSERT INTO registration_agency (agency_name, agency_address, contact_person, phone_number, email)
VALUES ('whatever', 'nowhere', 'someone', '123456789', '123@gmail.com');
```

c. edit / delete an institute

To edit the information of the institute, click the edit button next to it. Click again t o submit the changes. The front end will send the id (not displayed) of the institute and other information needed to the server. The same restriction in adding a new o ne applies here.

	名稱	地址	聯絡人	電話	email	距離 (km)
X done	whatever	nowhere	someone	123456789	123@gmail.com	N/A

Sample Update SQL

```
UPDATE registration_agency
SET agency_name = 'whatever',  agency_address = 'nowhere',  contact_person = 'someone',  phone_number = '123456789',  e
mail = '123@gmail.com'
WHERE agency_id = 1973;
```

Sample Delete SQL

```
DELETE FROM registration_agency
WHERE agency_id = 1973;
```

Problems & Solutions

1. Build the table of adopt animals : when we build the table of adopt animals, we find that th e data in it is not so unanimous. Many columns of data are in different format. We spend some time to correct the data type to solve the bug. Besides, the length of values in some columns change so much. In the beginning, we did not find it. We find it when we start to build the table in our database. We can not successfully read all the data into our table. We type the command "show warnings;", then we find this problem. We check those columns ro w by row. We find that there are some specific rows having bigger data than other rows. We adjust the length when we initialize our table then solve the problem.
2. Use the "Here map api": This api will return a list of the data near the address which we, because it can detect the exact address. As result we decide to use the data in the first element in the list.

Progress

Every thing above is completed. The application works without any error under exceptions.

However, there are still few things we want to achieve to make the App more comprehensive.

1. Auto update latitude and longitude

   To make the database more stable, auto update the lat and lng in registration_agency is important. Otherwise, we have convert the address into them manually for each insertion and update. But we met some problems in handling the invalid address. Sometimes even a valid address can be unidentifiable by the API.

2. A separate interface to manage institute information

   Currently, anyone can insert, edit and delete the data in registration_agency on the interface 相關機構. However, extra protection should have been provided. The new interface will be just like what we did in 收容所管理 and 飼主. The owner of the institutes have to log in before they can modify anything in the data.

3. More professional interfaces

   We did not spend too much time on UI and UX for they are not within the scope of the project. But if we want to promote the App in the future, the design of the interfaces have to be more sophisticated.

Contribution

0816203 陳永諭

1. Data
    - Web scraping

2. Proposal
    - Interfaces

3. Application
    - Front End
    - Back End
    - CRUD
    - Exception Handling

4. Presentation
    - Overal PPT design
    - “介面設計” in PPT
    - Demonstrate the Interfaces

5. Report
    - Application

0716089 王柏舜

1. Create table adopt_animal
2. Here api implement
3. query adopt animal and agency registration
4. Presentation
    - Spotlight video
5. Report
    - Maintain database
    - Backend process query
6. Proposal
    - App design

0816004 吳原博

1. Create table lost_pet
2. Conduct part of the normalization:
    - lost_pet_normalize and owner
    - Construct table shelter
3. Proposal
    - work schedule
4. Presentation
    - Dataset
    - Normalization
    - ER model
5. Report
    - Data

0816102 陳品戎

1. Create table registration_agency and shelter_information
2. Proposal
    - data column introduction
3. Csv file and create sql debug
4. Normalization
    - adopt_animal and registration_agency
5. Presentation
    - Datatset
6. Report
    - Data

Link to Trello board
https://trello.com/b/BX29f5ju/dbms-final-project

Link to Github Repo
https://github.com/yungyuchen521/DBMS-Final-Project