Main.py

```
# first line
# vet clinic emulator
import extentions.impslist as impslist

# variabili globali
generator = 1

################################################
#           Flow
def main():

#   crea lista per tutti gli owner e i pet
    impslist.ffile.initial_check()
    impslist.ffile.cls()
    print("\n\n_____________________________________")

    impslist.C.dynamic_pets(impslist.random.randint(1,21))
    impslist.C.dynamic_owners(impslist.random.randint(1,10))
    while True:
    #   sceglie se è il pet o l'owner o fare l'azione
        generator = impslist.random.randint(1,2)

        if generator == 1:
            #   PET CYCLE
            impslist.ffile.pet_cycle()
        elif generator == 2:
            #   OWNER CYCLE
            impslist.ffile.owner_cycle()

        impslist.time.sleep(impslist.random.uniform(0.02 , 2))

main()
# last  line
```

currently.py

```python
# responsable for creaing dynamic array of recovered pets and their owners
import objects.lista as Ls
import extentions.impslist as impslist

# lar = lista animali registrati
lar = Ls.PL
# lpr = lista proprietari registrati
lpr = Ls.OL

# liste dinamiche
animali = []
proprietari = []

# crea la lista di tutti gli animali presenti allinizio dato quack random nell'elenco di
animali totali
def dynamic_pets(quack):
    if quack > len(lar):
        print("error 3")
    print(f"there are currently {quack} pets in the clinic.")
    animali.extend(impslist.random.sample(lar, quack))
    Ls.print_List(animali, "pets")

def dynamic_owners(miao):
    if miao > len(lpr):
        print("error 8")
    print(f"there are currently {miao} owners registred in the clinic.")
    proprietari.extend(impslist.random.sample(lpr, miao))
    Ls.print_List(proprietari, "owners")

# last line
```

ffile.py

```python
import extentions.impslist as impslist
import extentions.currently as C

################################################
# generate a rondom time
def rtime():
  hour = impslist.random.randint(1,23)
  minute = impslist.random.randint(1,59)
  if hour > 9 and minute > 9:
    return print(f"{hour}:{minute}")
  elif hour < 9 and minute > 9:
    return print(f"0{hour}:{minute}")
  elif hour > 9 and minute < 9:
    return print(f"{hour}:0{minute}")
  else:
    return print(f"0{hour}:0{minute}")
################################################
# clean
def cls():
  impslist.time.sleep(2)
  impslist.os.system("cls" if impslist.os.name == "nt" else "clear")


################################################
# create a list with all pets and owners
def initial_check():
  impslist.Ls.owner_list()
  impslist.Ls.pets_list()

################################################
# restituisce direttamente l'oggetto Pet, non solo il nome
# gar da rivedere per fare una query SQL
def gar(obj_list, target_value, attr_name):
  for obj in obj_list:
    if getattr(obj, attr_name) == target_value:
      return obj
  raise ValueError(f"No object with {attr_name} = {target_value}")

################################################
def pet_cycle():

  new_action = impslist.random.randint(1,9)
  related_pet = []
  # avalable -> lista complementare a C.animali
  avalable_P = [pet for pet in impslist.Ls.PL if pet not in C.animali]

  if new_action in [1, 8]:
  # pesca da lista di non presenti
    if avalable_P:
      related_pet = impslist.random.choice(avalable_P)
      C.animali.append(related_pet)
      impslist.pet.action(new_action, related_pet)
```

```
  elif new_action in [2, 9]:
  #remove from list
    if not avalable_P:
      related_pet = impslist.random.sample(C.lar,1)[0]
      impslist.pet.action(new_action, related_pet)
      C.animali.remove(related_pet)

  elif new_action in [3, 4, 5, 6, 7]:
    related_pet = impslist.random.sample(C.animali, 1)[0]
    impslist.pet.action(new_action, related_pet)

  else:
    print("error 6")

#####################################################
def owner_cycle():

  new_action = impslist.random.randint(1,9)
  related_owner = []
  # avalable -> lista complementare a C.proprietari
  avalable_O = [owner for owner in impslist.Ls.OL if owner not in C.proprietari]

  if new_action in [3, 4, 10]:
  # pesca da lista di non presenti
    if avalable_O:
      related_owner = impslist.random.choice(avalable_O)
      C.proprietari.append(related_owner)
      impslist.owner.action(new_action, related_owner)

  elif new_action in [5, 11]:
  #remove from list
    if not avalable_O:
      related_owner = impslist.random.sample(C.lpr,1)[0]
      impslist.owner.action(new_action, related_owner)
      C.proprietari.remove(related_owner)

  elif new_action in [2, 6, 7, 8, 9]:
    related_owner = impslist.random.sample(C.proprietari, 1)[0]
    impslist.owner.action(new_action, related_owner)

  elif new_action == 1:
    related_owner = impslist.random.sample(C.lpr, 1)[0]
    impslist.owner.action(new_action, related_owner)
  else:
    print("error 8")

# last line
```

```
impslist.py
# IMPORTS LIST

#   PYTHON LIBRARIES
import os
import time
import random

#       FILE INTERI
import objects.entitties
import extentions.currently as C
import objects.lista as Ls
import extentions.ffile as ffile #function file

#       OBJS
import objects.pet as pet
import objects.owner as owner
from objects.owner import Owner
from objects.pet import Pet

# last line
```

entitties.py

```
# gli oggetti creatis
from objects.owner import Owner
from objects.pet import Pet

# owner -> 0 dipendenze (goes up)
owner_1 = Owner("Puffetto", "Spaziale", "MY050T1S", "14th of december", "M")
owner_2 = Owner("Moffettina", "Pinguinosa", "P4P4V3R1", "8th of october", "F")
owner_3 = Owner("Greta", "Crotone", "G1R450L1", "22th of november", "F")
owner_4 = Owner("Emily", "IDK", "C1CL4M1N1", "3th of genuary", "F")
owner_5 = Owner("Nicole", "Chinellato", "M4RGH3R1T4", "28th of april", "F")
owner_6 = Owner("Jacopo", "Moraldini", "L0T0814NC0", "2th of october", "M")
owner_7 = Owner("Luca", "Bertoldi", "P1L4V3L0", "15th of March", "M")
owner_8 = Owner("Giulia", "Verdi", "G4L123V", "22th of May", "F")
owner_9 = Owner("Stefano", "D'Ambrosio", "S3TA1C0", "12th of June", "M")
owner_10 = Owner("Francesca", "Giansanti", "F4NC3S9", "5th of July", "F")

# pet -> 1 dipendenza (goes down)
pet_1 = Pet("Pollosauro", 1.2, "120 g", "30 cm", "Nooty", "00001", "Healthy", "MY050T1S")
pet_2 = Pet("Sonnus canis", 20, "30 kg", "100 cm", "bianca", "00002", "Sonnite acuta",
"P4P4V3R1")
pet_3 = Pet("strambus selvaticus", 911, "84 kg", "69 cm", "weirdass thing", "0003",
"mentally ill owner", "G1R450L1")
pet_4 = Pet("Felis combustibilis", 3, "5.5 kg", "45 cm", "Zampa", "00004", "Flammable fur",
"C1CL4M1N1")
pet_5 = Pet("Canis depressus", 7, "20 kg", "80 cm", "Tristezza", "00005", "Existential
dread", "M4RGH3R1T4")
pet_6 = Pet("Capra lunatica", 5, "32 kg", "60 cm", "Becky la Svitata", "00006", "Mania
acuta", "L0T0814NC0")
pet_7 = Pet("Draconis minus", 0.4, "1.2 kg", "25 cm", "Pufflet", "00007", "Needs constant
supervision", "P4P4V3R1")
pet_8 = Pet("Tartaruga turbo", 80, "70 kg", "90 cm", "Nitro", "00008", "Hyperactive",
"M4RGH3R1T4")
pet_9 = Pet("Gatto quantico", 2, "3 kg", "40 cm", "Schrödy", "00009", "Alive and dead",
"MY050T1S")
pet_10 = Pet("Cane a molla", 1, "8 kg", "50 cm", "Slinky", "00010", "Joint issues",
"P4P4V3R1")
pet_11 = Pet("Coniglius paranoicus", 6, "4.2 kg", "30 cm", "Twitch", "00011", "Mild
conspiracy tendencies", "G1R450L1")
pet_12 = Pet("Topus gladiator", 0.9, "0.8 kg", "20 cm", "Spartacchio", "00012",
"Aggressively patriotic", "C1CL4M1N1")
pet_13 = Pet("Cacatua metallum", 10, "12 kg", "55 cm", "Rocky", "00013", "Addicted to heavy
metal", "M4RGH3R1T4")
pet_14 = Pet("Leone Ballerino", 200, "250 kg", "120 cm", "Rex", "00014", "Dancer at heart",
"P1L4V3L0")
pet_15 = Pet("Squalo Sognante", 1500, "800 kg", "6 m", "Sharky", "00015", "Dreams of the
ocean", "G4L123V")
pet_16 = Pet("Cervo Curioso", 20, "200 kg", "150 cm", "Bambi", "00016", "Loves to explore",
"S3TA1C0")
pet_17 = Pet("Cane Nube", 9, "20 kg", "65 cm", "Cloudy", "00017", "Is afraid of rain",
"F4NC3S9")
pet_18 = Pet("Gatto Cosmico", 4, "4 kg", "45 cm", "Nebula", "00018", "Not of this world",
"P1L4V3L0")
pet_19 = Pet("Tigre Elettrica", 300, "220 kg", "180 cm", "Volt", "00019", "Electrified
fur", "S3TA1C0")
```

```
pet_20 = Pet("Foca Acrobatica", 70, "100 kg", "150 cm", "Flippy", "00020", "Master of flips", "G4L123V")
pet_21 = Pet("peachy penguosus", 10, "2 kg", "40 cm", "pingu", "00021", "peachy", "P4P4V3R1")


# last line
```

illness.py

```python
# illness
import extentions.ffile as ffile

class Illness:
    def __init__(self, name, description, cure, lethality, life_cycles):
        # attributes -> what it is or has
        self.name = name
        self.description = description
        self.cure = cure
        self.lethality = lethality
        self.life_cycles = life_cycles

    def __str__(self):
        return f"Illness: {self.name} - Lethality: {'Yes' if self.lethality else 'No'}"

    # methods -> what it does

    def cure_treatment(self):
        if self.cure:
            print(f"Treatment for {self.name}: {self.cure}")
        else:
            print(f"No known cure for {self.name}")

    def disease_info(self):
        print(f"Description of {self.name}: {self.description}")

    def lifecycle_progress(self):
        if self.life_cycles > 0:
            print(f"{self.name} has {self.life_cycles} remaining life cycles.")
        else:
            print(f"{self.name} has no life cycles left and may need urgent attention!")

    def is_lethal(self):
        return self.lethality

# function to simulate action with illness
def action(illness, action_code):
    if action_code == 1:
        # Disease info
        illness.disease_info()
    elif action_code == 2:
        # Cure treatment
        illness.cure_treatment()
    elif action_code == 3:
        # Lifecycle status
        illness.lifecycle_progress()
    elif action_code == 4:
        # Check lethality
        if illness.is_lethal():
            print(f"{illness.name} is lethal!")
        else:
            print(f"{illness.name} is not lethal!")
```

```
  else:
    print("Invalid action code!")

# last line
```

lista.py

```
import objects.entitties as entitties

# lista di tutti i proprietari
OL = []
# lista di tutti i pet
PL = []

# crea la lista di tutti i proprietari esistenti
def owner_list():
  for name in dir(entitties):
    quack = getattr(entitties, name)
    if isinstance(quack, entitties.Owner):
      OL.append(quack)
#    print_List(OL, "owners")

# crea la lista di tutti gli animali esistenti
def pets_list():
  for name in dir(entitties):
    noot = getattr(entitties, name)
    if isinstance(noot, entitties.Pet):
      PL.append(noot)
#    print_list(PL, "pets")

# stampa la lista degli
#   which -> owner o pet
def print_List(n, which):
  print(f"the list of {which}:\n")
  x = 0
  while x != len(n):
    print(f"{x +1}. ", end="")
    print(n[x])
    x += 1
  print("")

# last line
```

owner.py

```
# owner
import extentions.ffile as ffile

class Owner:
  def __init__(self, name, surname, CF, bday, gender):
    self.name = name
    self.surname = surname
    self.CF = CF
    self.bday = bday
    self.gender = gender

  def __str__(self):
    return f"{self.CF}"

  def birthday(self):
    if self.gender == 'M':
      print(f"mr. {self.surname}'s birthday is {self.bday}")
    elif self.gender == 'F':
      print(f"miss {self.surname}'s birthday is {self.bday}")
    else:
      print(f"{self.surname} the fuck is wrong with you? get out of here!\n*grabs a shotgun*")
      print(f"*the assistant throws at {self.name} it's pet*")

  def check(self, pet):
    print(f"{self.surname} called to check on {pet} at ", end="")
    ffile.rtime()

  def pav(self):
    print(f"{self.surname} booked a vet visit for ", end="")
    ffile.rtime()

  def register_pet(self, pet):
    print(f"{self.surname} registered {pet} at ", end="")
    ffile.rtime()

  def report_death(self, pet):
    print(f"{self.surname} reported that {pet} has died at ", end="")
    ffile.rtime()

  def feed_pet(self, pet):
    print(f"{self.surname} is feeding {pet} at ", end="")
    ffile.rtime()

  def walk_pet(self, pet):
    print(f"{self.surname} is walking {pet} around the block")

  def play_with_pet(self, pet):
    print(f"{self.surname} is playing with {pet} — happy tails and wagging guaranteed")

  def judge_pet(self, pet):
    print(f"{self.surname} is judging {pet}'s behavior silently... but deeply.")
```

```
  def adopt_pet(self, pet):
    print(f"{self.surname} adopted {pet} from the clinic at ", end="")
    ffile.rtime()

  def abandon_pet(self, pet):
    print(f"{self.surname} abandonet {pet} at the clinic at ", end="")
    ffile.rtime()

# usare la funzione ffile.get_by_attribute(obj_list, target_value, attr_name) per trovareil
nome del pet relativo

def action(code, owner):
  from objects.lista import PL
  # l'idea è di prendere il codice dell'animale relativo al proprietario
  pet = ffile.gar(PL, owner.CF, "owner")

  if code == 1:
    owner.birthday()
  elif code == 2:
    owner.check(pet)
  elif code == 3:
    owner.pav()
  elif code == 4:
    owner.register_pet(pet)
  elif code == 5:
    owner.report_death(pet)
  elif code == 6:
    owner.feed_pet(pet)
  elif code == 7:
    owner.walk_pet(pet)
  elif code == 8:
    owner.play_with_pet(pet)
  elif code == 9:
    owner.judge_pet(pet)
  elif code == 10:
    owner.adopt_pet(pet)
  elif code == 11:
    owner.abandon_pet(pet)
  else:
    print("error 7")


# last line
```

pet.py

```python
# pet
import extentions.ffile as ffile

class Pet:
  # INIT METHOD COSTRUISCE I METODI -> __init__(self):

  def __init__(self, species, age, weight, height, name, id, condition, owner):
    # attributes -> what it is or has
    self.species = species
    self.age = age
    self.weight = weight
    self.height = height
    self.name = name
    self.condition = condition
    self.id = id
    self.owner = owner

  def __str__(self):
    return f"{self.id}" # -> owner is: {self.owner}"

  # methods -> what it does

  def birth(self):

    print(f"{self.name} has born at ", end="")
    ffile.rtime()

  def death(self):
    print(f"{self.name} has died at ", end="")
    ffile.rtime()

  def eat(self):
    print(f"{self.name} is eating at ", end="")
    ffile.rtime()

  def move(self):
    print(f"{self.name} is moving according to what being {self.species} allows")

  def play(self):
    print(f"{self.name} loves to play with {self.owner}")

  def sleep(self):
    print(f"{self.name} has been sleeping since: ", end="")
    ffile.rtime()

  def judge(self):
    print(f"{self.name} is judging you")

  def admitted(self):
    print(f"{self.name} has been admitted to the clinic at ", end="")
    ffile.rtime()
```

```
  def dismissed(self):
    print(f"{self.name} has been dismissedd from the clinc at ", end="")
    ffile.rtime()

def action(miao, pet):
  if miao == 1:
    # Caso 1: far nascere l'animale
    pet.birth()

  elif miao == 2:
    # Caso 2: l'animale è morto
    pet.death()

  elif miao == 3:
    # Caso 3: l'animale mangia
    pet.eat()

  elif miao == 4:
    # Caso 4: l'animale si muove
    pet.move()

  elif miao == 5:
    # Caso 5: l'animale gioca
    pet.play()

  elif miao == 6:
    # Caso 6: l'animale dorme
    pet.sleep()

  elif miao == 7:
    # Caso 7: l'animale ti giudica
    pet.judge()

  elif miao == 8:
    # Caso 8: l'animale viene ammesso in clinica
    pet.admitted()

  elif miao == 9:
    # Caso 9: l'animale viene dimesso dalla clinica
    pet.dismissed()

  else:
    # Caso di errore se miao non è valido
    print("error 5!")

# last line
```

connection.py

```
# first line

import time
import os
import psycopg2
import json
from tabulate import tabulate

def close(cur, cnt):
    cur.close()
    cnt.close()

# print table
def tprint (cur, tname):
    # per stampare i risultati
    try:
        # Esegui la query per ottenere tutti i dati dalla tabella
        cur.execute(f"SELECT * FROM {tname};")
        rows = cur.fetchall()
        if rows:
            # Stampa la tabella con `tabulate`
            print(tabulate(rows, headers=[], tablefmt="psql"))
        else:
            print(f"No data found in table {tname}.")
    except Exception as e:
        print(f"Error fetching data from table {tname}: {str(e)}")

#    create the db - just a bunch of empty tables
def make_db(cur):
    #    apre il file in lettura e copia il contenuto come fosse una stringa
    with open('db.sql') as file:
        db = file.read()
    #    passa la stringa come comando
    cur.execute(db)
    print("tables created, now gotta populate 'em")

# Connects with the server
def connect():
    # Connessione al server di PostgreSQL

    # Open the credentials file and load the JSON data
    with open("credentials.json", "r") as nooty:
        pruty = json.load(nooty)
    try:
        # ** -> scompone il dict in parametri
        conn = psycopg2.connect(**pruty)
        return conn
    except Exception as e:
        print("Failed to establish connection:\n", e)
        return None

#    first interation with the db
```

```
def first_connection():

  cnt = connect()
  if cnt is None:
    print ("couldn't connect to db or somme other shit you gotta check.\nfile is
connection.py")
    return
  else:
    print ("Connection established correctly, yay!")
    cur = cnt.cursor()
    make_db(cur)
    # commit() è un modulo  di connect
    cnt.commit()
  return cnt, cur

#   fills the db's tables
def populate(cnt, cur):
  def fill_query(miao, cur):
    if miao == 0:
      print("starting loop")
      return 1

    if miao == 1:
      # tabella owner
      try:
        with open("owner tablefill.sql") as file:
          quack = file.read()
          quack = str(quack)
          cur.execute(quack)
        file.close()
      except Exception as e:
        print("error 9.\n " + str(e))
        return None
      else:
        print("\nowner table... done")

        time.sleep(2)
        os.system("clear")
        tprint(cur, "owners")
        cnt.commit()
        return "quack"

    elif miao == 2:
      # tabella illness
      try:
        with open("illness tablefill.sql") as file:
          peachy = file.read()
          peachy = str(peachy)
          cur.execute(peachy)
        file.close()
      except Exception as e:
        print("error 10.\n " + str(e))
```

```
            return None
        else:
            print("\nillness table... done")

            time.sleep(2)
            os.system("clear")
            tprint(cur, "illness")
            cnt.commit()
            return "pru"

    elif miao == 3:
        # tabella pet
        try:
            with open("pets tablefill.sql") as file:
                lemon = file.read()
                lemon = str(lemon)
                cur.execute(lemon)
            file.close()
        except Exception as e:
            print("error 11.\n " + str(e))
            return None
        else:
            print("\npets table... done")

            time.sleep(2)
            os.system("clear")
            tprint(cur, "pets")
            cnt.commit()
            return "miao"

    print("we're populating")

    bl = 0
    while bl in [0, 1, 2, 3]:
        checking = fill_query(bl, cur)
        bl +=1
        if checking is None:
            return

# flow
def flow():
    connessione, cursore = first_connection()
    cur = populate(connessione, cursore)
    time.sleep(2)
    os.system("clear")

flow()
# last line
```

credentials.json

```
{
   "dbname": "vet clinic",
   "user" : "postgres",
   "password": "160718",
   "host": "localhost",
   "port": "5432"
}
```

db.sql

```
-- first line

--      table for owners

CREATE TABLE IF NOT EXISTS owners (

  CF VARCHAR(20) PRIMARY KEY,
  name VARCHAR(20),
  surname VARCHAR(20),
  birthday DATE,
  gender VARCHAR(1)
);

--          table for the illness
CREATE TABLE IF NOT EXISTS illness (

  code SERIAL PRIMARY KEY,
  name VARCHAR(50),
  description VARCHAR(100),
  cure VARCHAR(50) NULL,
  lethality BOOLEAN,
  life_cycles INT

);

--      table for the pets

CREATE TABLE IF NOT EXISTS pets (
  id SERIAL PRIMARY KEY,
  name VARCHAR(20),
  species VARCHAR(40),
  age FLOAT,
  peso FLOAT,
  height FLOAT,
  birth DATE NULL,
  death DATE NULL,
  stato BOOLEAN,
  condition INT REFERENCES illness(code) NULL,
  owner_id VARCHAR(20) REFERENCES owners(CF) NULL

);


-- last line
```

```
illness tablefill.sql
-- first line

-- si usa una query per riempire la tabella con gli obj illness

-- Inserimento delle malattie con i dati richiesti
;
INSERT INTO illness(name, description, cure, lethality, life_cycles)
VALUES
  ('Sonnite acuta', 'Infiammazione acuta delle vie respiratorie', 'Riposo e antibiotici', FALSE, 1),
  ('Flammable fur', 'Pelo altamente infiammabile', 'Controllo ambientale', TRUE, 2),
  ('Existential dread', 'Ansia esistenziale', 'Terapia cognitivo-comportamentale', FALSE, 3),
  ('Mania acuta', 'Disturbo mentale con sintomi di esaltazione e impulsività', 'Farmaci stabilizzatori dell’umore', TRUE, 4),
  ('Needs constant supervision', 'Necessità di sorveglianza continua', 'Monitoraggio costante', FALSE, 5),
  ('Hyperactive', 'Comportamento iperattivo', 'Controllo comportamentale', FALSE, 2),
  ('Alive and dead', 'Condizione di apparente morte temporanea', 'Monitoraggio medico', TRUE, 6),
  ('Joint issues', 'Problemi alle articolazioni', 'Farmaci antinfiammatori', FALSE, 3),
  ('Mild conspiracy tendencies', 'Tendenze paranoiche e sospettose', 'Terapia psicologica', FALSE, 1),
  ('Aggressively patriotic', 'Comportamento estremamente patriottico', 'Supporto psicologico', FALSE, 2),
  ('Addicted to heavy metal', 'Dipendenza da musica heavy metal', 'Terapia di disintossicazione musicale', FALSE, 3),
  ('Dancer at heart', 'Amore per la danza', 'Nessuna cura necessaria', FALSE, 4),
  ('Dreams of the ocean', 'vuole vivere nel mare', 'Supporto psicologico', FALSE, 2),
  ('Loves to explore', 'Ama esplorare', 'Nessuna cura necessaria', FALSE, 5),
  ('Is afraid of rain', 'Paura intensa della pioggia', 'Terapia psicologica', FALSE, 2),
  ('Not of this world', 'Comportamento alienante e distacco dalla realtà', 'Supporto psicologico', TRUE, 7),
  ('Electrified fur', 'Pelo che genera scariche elettriche', 'Isolamento dal campo elettrico', TRUE, 2),
  ('Master of flips', 'Capacità di eseguire acrobazie in modo perfetto', 'Nessuna cura necessaria', FALSE, 3),
  ('peachy', 'Condizione psicologica di benessere e tranquillità', 'Relax e meditazione', FALSE, 1)

ON CONFLICT(code) DO NOTHING

-- last line
```

owner tablefill.sql

```sql
-- si crea una query per riempire le tabelle con tutti gli owner
INSERT INTO owners(name, surname, CF, birthday, gender)
VALUES
  ('Puffetto', 'Spaziale', 'MY050T1S', '2002-12-14', 't'),
  ('Moffettina', 'Pinguinosa', 'P4P4V3R1', '1995-10-08', 'f'),
  ('Greta', 'Crotone', 'G1R450L1', '2000-11-22', 'f'),
  ('Emily', 'IDK', 'C1CL4M1N1', '1998-01-03', 'f'),
  ('Nicole', 'Chinellato', 'M4RGH3R1T4', '1990-04-28', 'f'),
  ('Jacopo', 'Moraldini', 'L0T0814NC0', '2002-10-02', 't'),
  ('Luca', 'Bertoldi', 'P1L4V3L0', '1989-03-15', 't'),
  ('Giulia', 'Verdi', 'G4L123V', '1995-05-22', 'f'),
  ('Stefano', 'Dambrosio', 'S3TA1C0', '1988-06-12', 't'),
  ('Francesca', 'Giansanti', 'F4NC3S9', '1997-07-05', 'f')
ON CONFLICT (CF) DO NOTHING;
-- last line
```

pets tablefill.sql

```sql
-- first line

INSERT INTO pets(name, species, age, peso, height, stato, owner_id)
VALUES
  ('Pollosauro', 'Nooty', 1.2, 120, 30, 't', 'MY050T1S'),
  ('Sonnus canis', 'bianca', 20, 30, 100, 'f', 'P4P4V3R1'),
  ('strambus selvaticus', 'weirdass thing', 911, 84, 69, 't', 'G1R450L1'),
  ('Felis combustibilis', 'Zampa', 3, 5.5, 45, 'f', 'C1CL4M1N1'),
  ('Canis depressus', 'Tristezza', 7, 20, 80, 'f', 'M4RGH3R1T4'),
  ('Capra lunatica', 'Becky la Svitata', 5, 32, 60, 'f', 'L0T0814NC0'),
  ('Draconis minus', 'Pufflet', 0.4, 1.2, 25, 'f', 'P4P4V3R1'),
  ('Tartaruga turbo', 'Nitro', 80, 70, 90, 'f', 'M4RGH3R1T4'),
  ('Gatto quantico', 'Schrödy', 2, 3, 40, 't', 'MY050T1S'),
  ('Cane a molla', 'Slinky', 1, 8, 50, 'f', 'P4P4V3R1'),
  ('Coniglius paranoicus', 'Twitch', 6, 4.2, 30, 't', 'G1R450L1'),
  ('Topus gladiator', 'Spartacchio', 0.9, 0.8, 20, 't', 'C1CL4M1N1'),
  ('Cacatua metallum', 'Rocky', 10, 12, 55, 't', 'M4RGH3R1T4'),
  ('Leone Ballerino', 'Rex', 200, 250, 120, 't', 'P1L4V3L0'),
  ('Squalo Sognante', 'Sharky', 1500, 800, 600, 't', 'G4L123V'),
  ('Cervo Curioso', 'Bambi', 20, 200, 150, 't', 'S3TA1C0'),
  ('Cane Nube', 'Cloudy', 9, 20, 65, 't', 'F4NC3S9'),
  ('Gatto Cosmico', 'Nebula', 4, 4, 45, 't', 'P1L4V3L0'),
  ('Tigre Elettrica', 'Volt', 300, 220, 180, 't', 'S3TA1C0'),
  ('Foca Acrobatica', 'Flippy', 70, 100, 150, 't', 'G4L123V'),
  ('peachy penguosus', 'pingu', 10, 2, 40, 'f', 'P4P4V3R1')

ON CONFLICT (id) DO NOTHING


-- last line
```

Schema.puml

```
@startuml Vet_Clinic_Flow

left to right direction
skinparam packageStyle rectangle

package "Main Flow" {
  [main.py] --> [impslist.py] : import
  [main.py] --> [ffile.py] : usa
  [main.py] --> [connection.py] : setup DB
}

package "Execution Logic" {
  [ffile.py] --> [currently.py] : usa
  [currently.py] --> [lista.py] : usa liste
  [lista.py] --> [entitties.py] : importa oggetti
  [entitties.py] --> [owner.py] : istanzia Owner
  [entitties.py] --> [pet.py] : istanzia Pet
}

[ffile.py] --> [pet.py] : pet_cycle()
[ffile.py] --> [owner.py] : owner_cycle()

package "Objects" {
  [owner.py] --> [pet.py] : possiede (via CF)
  [pet.py] --> [illness.py] : ha condizione
}

package "Database" {
  [connection.py] --> [db.sql] : crea tabelle
  [connection.py] --> [owner tablefill.sql] : riempie
  [connection.py] --> [pets tablefill.sql] : riempie
  [connection.py] --> [illness tablefill.sql] : riempie
}

@enduml
```