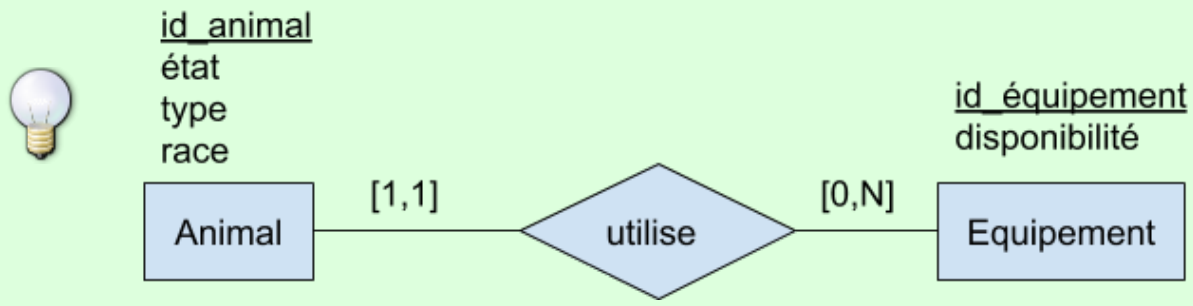
TD4 : Création d'un schéma de données

- Le but est maintenant de définir un modèle ORM pour le schéma de données du TD1. Reprenez votre programme de gestion de l'animalerie (TD 1 et 2). Le but est de remplacer les fichiers json par une base de données sqlite en utilisant les fonctionnalités de pony pour lire et mettre à jour les données.
- La première étape consiste à définir le modèle de données. Pour conserver les données de l'animalerie, on utilisera le gestionnaire de bases de données `sqlite`. Avant toute chose, il faut définir un schéma de données conforme au modèle relationnel

On part du schéma entité/association suivant:



A FAIRE :

- définissez le schéma relationnel correspondant (sans oublier les clés étrangères)
- traduisez le schéma relationnel en schéma UML
- Ajoutez le nouveau script `data_model.py` à votre projet, et définissez le schéma de données à l'aide des fonctions de pony:

```
from pony import orm

db = orm.Database()

class Equipement(db.Entity):
    ...

class Animal(db.Entity):
    ...
```

- Vous devez maintenant remplir la base à l'aide des données contenues dans animal.json et équipement.json. Pour ce faire, utilisez le script suivant (il ne devra être exécuté qu**une seule fois**).

```
import json

from pony import orm
from data_model import Equipement, Animal, db

db.bind(provider='sqlite', filename='animalerie.db', create_db=True)
```

```
db.generate_mapping(create_tables=True)

équipement_data = 'équipement.json'
with open(équipement_data, "r") as f:
    équipement_dict = json.load(f)
    for id_équip in équipement_dict:
        disponibilité = équipement_dict[id_équip]["DISPONIBILITÉ"]
        with orm.db_session:
            try:
                Equipement(id_équip=id_équip, disponibilité=disponibilité)
                orm.commit()
            except:
                print(id_équip, "already exists in database")
                pass


animal_data = 'animal.json'
with open(animal_data, "r") as f:
    animal_dict = json.load(f)
    for id_animal in animal_dict:
        état = animal_dict[id_animal]["ETAT"]
        type = animal_dict[id_animal]["TYPE"]
        race = animal_dict[id_animal]["RACE"]
        lieu = animal_dict[id_animal]["LIEU"]
        with orm.db_session:
            try:
                Animal(id_animal=id_animal,
                       état=état,
                       type=type,
                       race=race,
                       lieu=Equipement[lieu])
                orm.commit()
            except:
                print(id_animal, "already exists in database")
                pass
```

- Vous disposez maintenant d'une base de données `animalerie.db` dans le répertoire du projet. Cette base contient l'ensemble des informations nécessaires pour gérer l'animalerie. Vous devez maintenant reprendre votre programme de gestion de l'animalerie (TD 1 et 2) et modifier `modele.py` en utilisant les fonctionnalités de `pony` pour lire et mettre à jour les données.

Voici à quoi doit ressembler le début de `modele.py` :

```
from pony import orm
from data_model import Equipement, Animal, db

liste_états = ['affamé', 'fatigué', 'repus', 'endormi']

db.bind(provider='sqlite', filename='animalerie.db')
db.generate_mapping()
```

```
def lit_état(id_animal):
    with orm.db_session:
        try:
            return Animal[id_animal].état
        except:
            return None

def lit_lieu(id_animal):
    with orm.db_session:
        try:
            return Animal[id_animal].lieu
        except:
            return None

def vérifie_disponibilité(id_équipement):
    ...
```

Complétez le code de manière à valider le fichier de tests suivant :

```
import modele
import controleur
from data_model import orm, Equipement, Animal

def test_lit_etat():
    assert modele.lit_état('Tac') == 'affamé'
    assert modele.lit_état('Bob') == None

@orm.db_session
def test_lit_lieu():
    assert modele.lit_lieu('Tac') == Equipement['litière']
    assert modele.lit_lieu('Bob') == None

def test_vérifie_disponibilité():
    assert modele.vérifie_disponibilité('litière') == 'libre'
    assert modele.vérifie_disponibilité('roue') == 'occupé'
    assert modele.vérifie_disponibilité('nintendo') == None

@orm.db_session
def test_cherche_occupant():
    assert Animal['Totoro'] in modele.cherche_occupant('roue')
    assert Animal['Tac'] in modele.cherche_occupant('litière')
    assert Animal['Tac'] not in modele.cherche_occupant('mangeoire')
    assert modele.cherche_occupant('nintendo') == []

def test_change_état():
    modele.change_état('Totoro', 'fatigué')
    assert modele.lit_état('Totoro') == 'fatigué'
    modele.change_état('Totoro', 'excité comme un pou')
    assert modele.lit_état('Totoro') == 'fatigué'
    modele.change_état('Truc', 'fatigué')
```

```
    assert modele.lit_état('Truc') == None

@orm.db_session
def test_change_lieu():
    modele.change_lieu('Totoro', 'roue')
    assert modele.lit_lieu('Totoro') == Equipement['roue']
    modele.change_lieu('Totoro', 'nid')
    assert modele.lit_lieu('Totoro') == Equipement['roue']
    modele.change_lieu('Totoro', 'nintendo')
    assert modele.lit_lieu('Totoro') == Equipement['roue']
    modele.change_lieu('Muche', 'litière')
    assert modele.lit_lieu('Muche') == None



@orm.db_session
def test_nourrir():
    if modele.vérifie_disponibilité('mangeoire') == 'libre' and
modele.lit_état('Tic') == 'affamé':
        controleur.nourrir('Tic')
    assert modele.vérifie_disponibilité('mangeoire') == 'occupé'
    assert modele.lit_état('Tic') == 'repus'
    assert modele.lit_lieu('Tic') == Equipement['mangeoire']
    controleur.nourrir('Pocahontas')
    assert modele.lit_état('Pocahontas') == 'endormi'
    assert modele.lit_lieu('Pocahontas') == Equipement['nid']
    controleur.nourrir('Tac')
    assert modele.lit_état('Tac') == 'affamé'
    assert modele.lit_lieu('Tac') == Equipement['litière']
    controleur.nourrir('Bob')
    assert modele.lit_état('Bob') == None
    assert modele.lit_lieu('Bob') == None
    assert modele.vérifie_disponibilité('mangeoire') == 'occupé'
```

Vérifiez également que votre programme fonctionne correctement avec l'interface graphique définie au TD2