



Penser les données par un ORM: les modèles



Bienvenue sur l'application du Factbook!

Voici le lien vers le site officiel : <https://www.cia.gov/the-world-factbook/>

| # | Nom | Capitale | Continent |
|----|--|----------|-----------|
| 1 | Algeria | inconnu | inconnu |
| 2 | Angola | inconnu | inconnu |
| 3 | Botswana | inconnu | inconnu |
| 4 | Benin | inconnu | inconnu |
| 5 | Burundi | inconnu | inconnu |
| 6 | Chad | inconnu | inconnu |
| 7 | Congo | inconnu | inconnu |
| 8 | Congo DR | inconnu | inconnu |
| 9 | Cameroon | inconnu | inconnu |
| 10 | Comoros | inconnu | inconnu |
| 11 | Central African Republic | inconnu | inconnu |

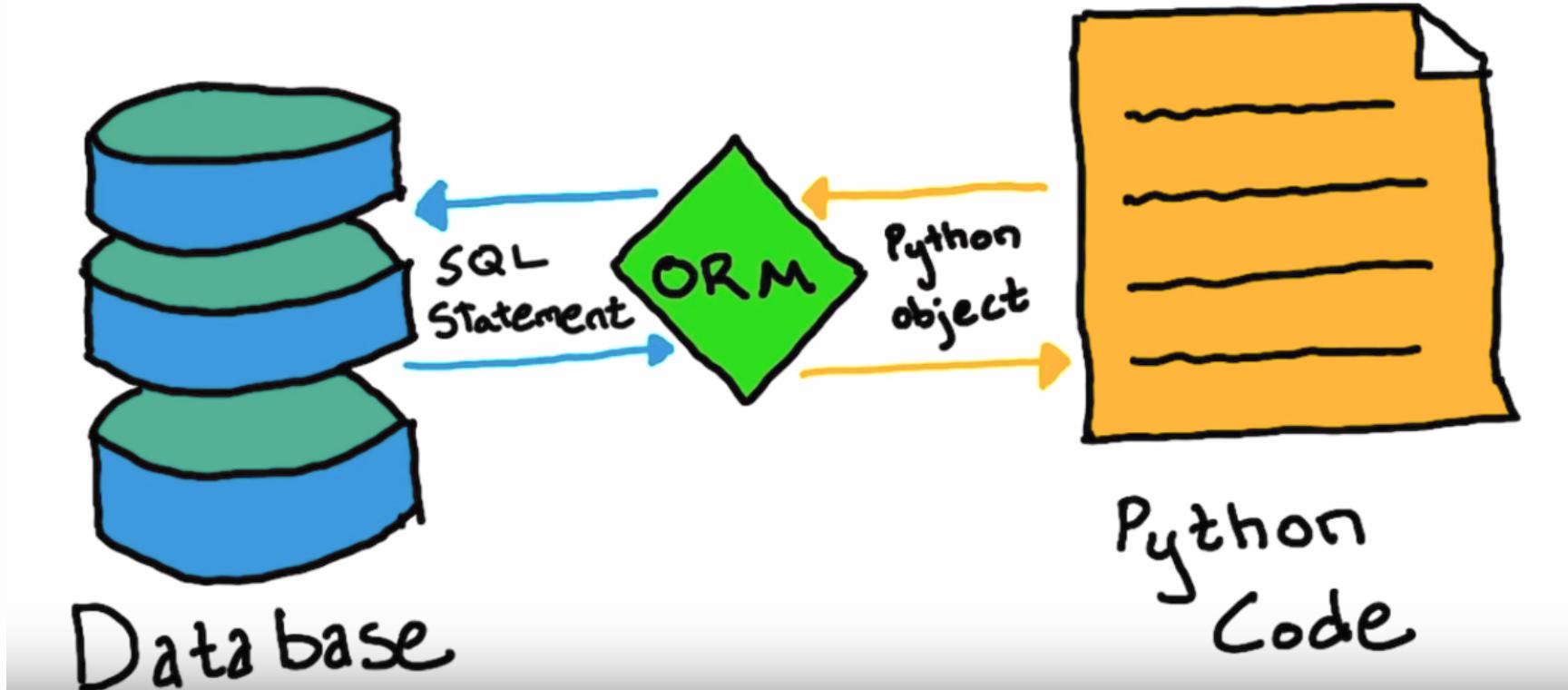
Installation de SQLite3

```
#! Linux  
sudo apt-get install sqlite3
```

```
#! Windows  
#! télécharger https://www.sqlite.org/download.html  
#! exécuter sqlite3.exe
```

```
#! Mac  
brew install sqlite3
```

Les Object-Relational Mapping (ORM)



Flask-SQLAlchemy

```
#! dans l'environnement virtuel  
  
pip install flask_sqlalchemy
```

```
...  
Collecting SQLAlchemy>=1.4.18  
Collecting Werkzeug>=2.2.2  
Collecting Jinja2>=3.0  
...
```

Connecter la base de données

- Ajout d'une variable d'environnement

```
# .env

# chemins relatifs (et absolu sous Windows)
SQLALCHEMY_DATABASE_URI.sqlite:///...

# chemins absolus sous Linux et Mac
SQLALCHEMY_DATABASE_URI.sqlite:///...

# pour les autres SGBD
SQLALCHEMY_DATABASE_URI=[DB_TYPE]+[DB_CONNECTOR]:///[USERNAME]::
[PASSWORD]@[HOST]:[PORT]/[DB_NAME]
```

Connecter la base de données

- Ajout de la variable dans `Config()`

```
# config.py

class Config():
    DEBUG = os.environ.get("DEBUG")
    SQLALCHEMY_DATABASE_URI = os.environ.get("SQLALCHEMY_DATABASE_URI")
```

Connecter la base de données

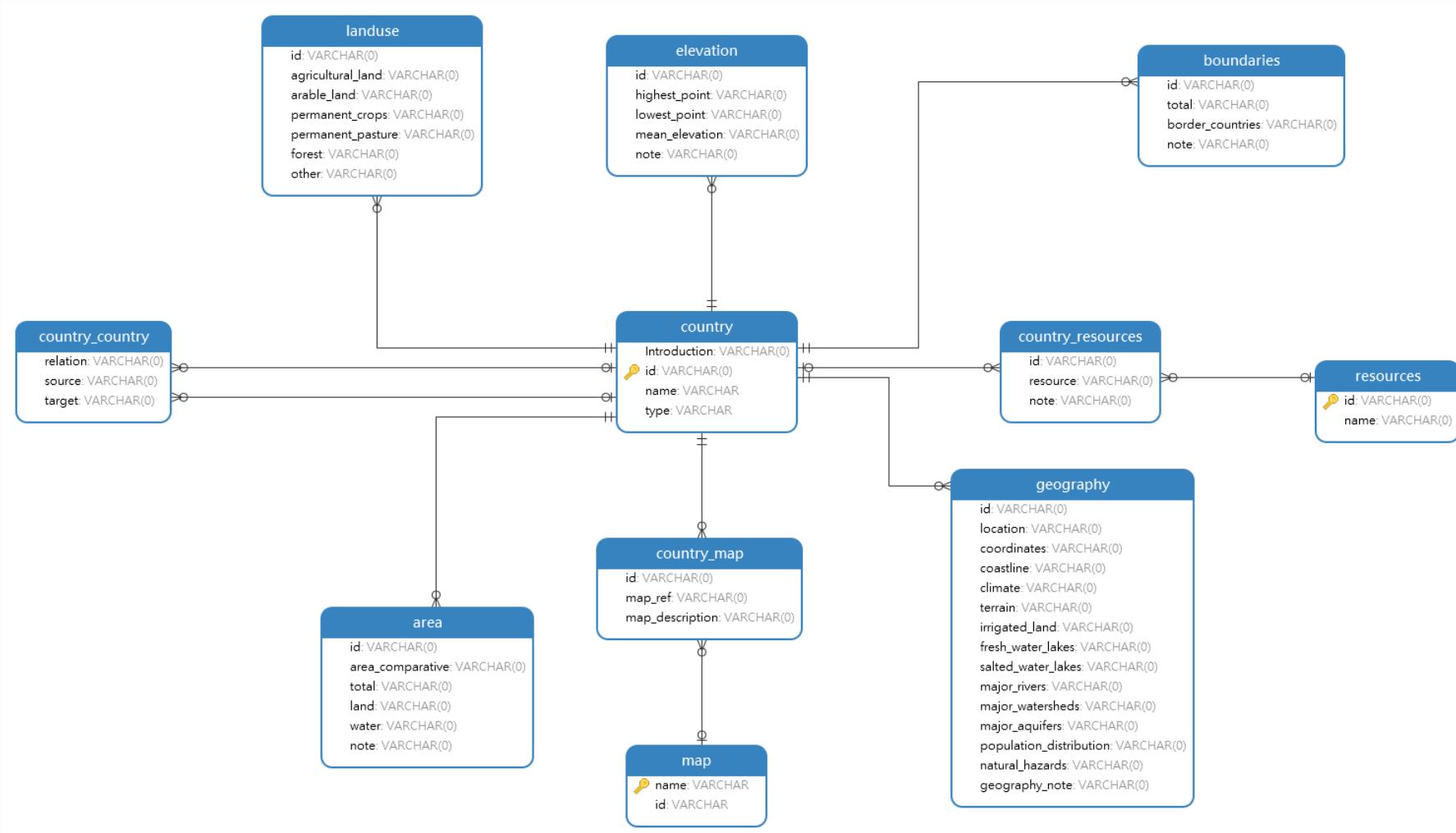
- Instanciation de Flask-SQLAlchemy

```
#app.py

...
from flask_sqlalchemy import SQLAlchemy
...
# app.config['SQLALCHEMY_DATABASE_URI'] = Config().SQLALCHEMY_DATABASE_URI

db = SQLAlchemy(app)
...
from .routes import generales
```

Modèle de données physique



Une requête pour s'assurer de la bonne connection

```
# app/generales.py

...
@app.route("/pays")
def pays():
    ...
    print(db.session.execute("SELECT * FROM country LIMIT 2").fetchall())
    return render_template("pages/tous_pays.html",
        donnees=donnees,
        sous_titre="Tous les pays")
```

Résultat attendu

```
[("p>Algeria has known many empires and dynasties starting with the ancient Numidians (3rd century B.C.), Phoenicians, Carthaginians, Romans, Vandals, ... (2790 characters truncated) ... arbon revenues to fund the government and finance the large subsidies for the population has fallen under stress because of declining oil prices.</p>", 'ag', 'Algeria', 'sovereign'), ("From the late 14th to the mid 19th century a Kingdom of Kongo stretched across central Africa from present-day northern Angola into the current Congo ... (1141 characters truncated) ... He pushed through a new constitution in 2010. Joao LOURENCO was elected president in August 2017 and became president of the MPLA in September 2018.", 'ao', 'Angola', 'sovereign')]  
127.0.0.1 - - [27/Nov/2022 17:18:40] "GET /pays HTTP/1.1" 200 -
```

Les modèles de base de données

Un premier modèle

```
# models/factbook.py
from ..app import app, db

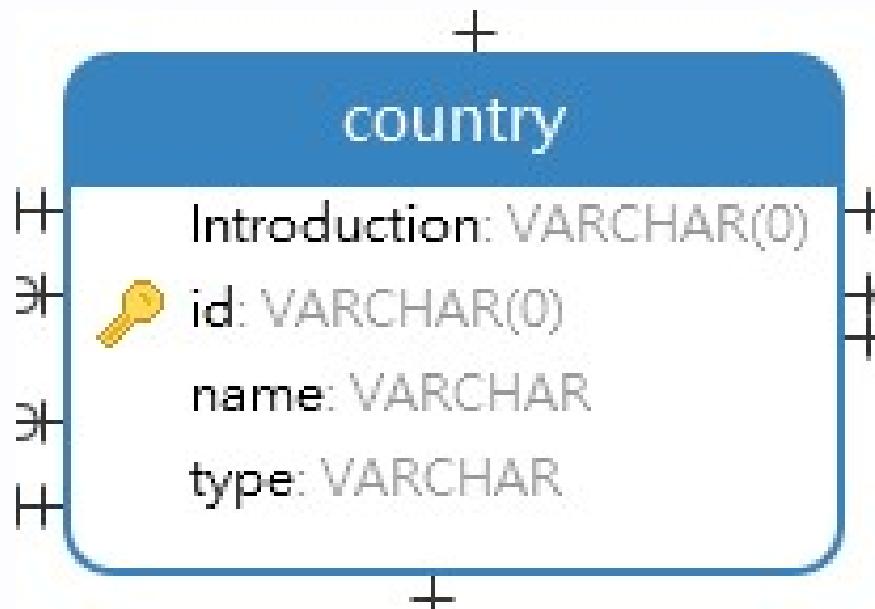
class Country(db.Model):
    ...
```

```
# syntaxe générale

nom = db.Column(type de colonne, paramètres supplémentaires)
```

Exercice

A partir de la syntaxe précédente et de la documentation de Flask-SQLAlchemy, écrire le modèle de la table `country`



Correction

```
# models/factbook.py

from ..app import app, db

class Country(db.Model):
    id = db.Column(db.String(10))
    Introduction = db.Column(db.Text)
    name = db.Column(db.String(500))
    type = db.Column(db.String(100))
```

Poser des contraintes

- Clé primaire

```
# models/factbook.py

class Country(db.Model):
    id = db.Column(db.String(10), primary_key=True)
    Introduction = db.Column(db.Text)
    name = db.Column(db.String(500))
    type = db.Column(db.String(100))
```

Poser des contraintes

- Unicité

```
# models/factbook.py

class Country(db.Model):
    id = db.Column(db.String(10), primary_key=True)
    Introduction = db.Column(db.Text)
    name = db.Column(db.String(500), unique=True)
    type = db.Column(db.String(100))
```

Poser des contraintes

- NOT NULL

```
# models/factbook.py

class Country(db.Model):
    id = db.Column(db.String(10), primary_key=True)
    Introduction = db.Column(db.Text)
    name = db.Column(db.String(500), unique=True, nullable=False)
    type = db.Column(db.String(100))
```

Compléments

- le `__tablename__`

```
# models/factbook.py

class Country(db.Model):
    __tablename__ = "country"
    ...
```

- les `__table_args__`

```
class Country(db.Model):
    __table_args__ = {"mysql_engine": "TokuDB"}
    __table_args__ = {"schema": "nom_du_schema"}
```

Compléments

- la méthode `__repr__()`

```
# models/factbook.py

class Country(db.Model):
    __tablename__ = "country"

    id = db.Column(db.String(10), primary_key=True)
    Introduction = db.Column(db.Text)
    name = db.Column(db.String(500), unique=True, nullable=False)
    type = db.Column(db.String(100))

    def __repr__(self):
        return '<Country %r>' % (self.name)
```

Exercice

Dans la route `/pays`, afficher tous les pays présents dans la base de données. Quelques étapes:

- interroger les données via l'ORM grâce à `Country.query.all()`
- comprendre ce que renvoie cette requête
- transformer les résultats remontés pour construire les données dans le format nécessaire au template de `/pays`

Penser les données par un ORM: les modèles

Résultat attendu



Bienvenue sur l'application du Factbook!

Voici le lien vers le site officiel : <https://www.cia.gov/the-world-factbook/>

| # | Nom | Capitale | Continent |
|----|--|----------|-----------|
| 1 | Algeria | inconnu | inconnu |
| 2 | Angola | inconnu | inconnu |
| 3 | Botswana | inconnu | inconnu |
| 4 | Benin | inconnu | inconnu |
| 5 | Burundi | inconnu | inconnu |
| 6 | Chad | inconnu | inconnu |
| 7 | Congo | inconnu | inconnu |
| 8 | Congo DR | inconnu | inconnu |
| 9 | Cameroon | inconnu | inconnu |
| 10 | Comoros | inconnu | inconnu |
| 11 | Central African Republic | inconnu | inconnu |

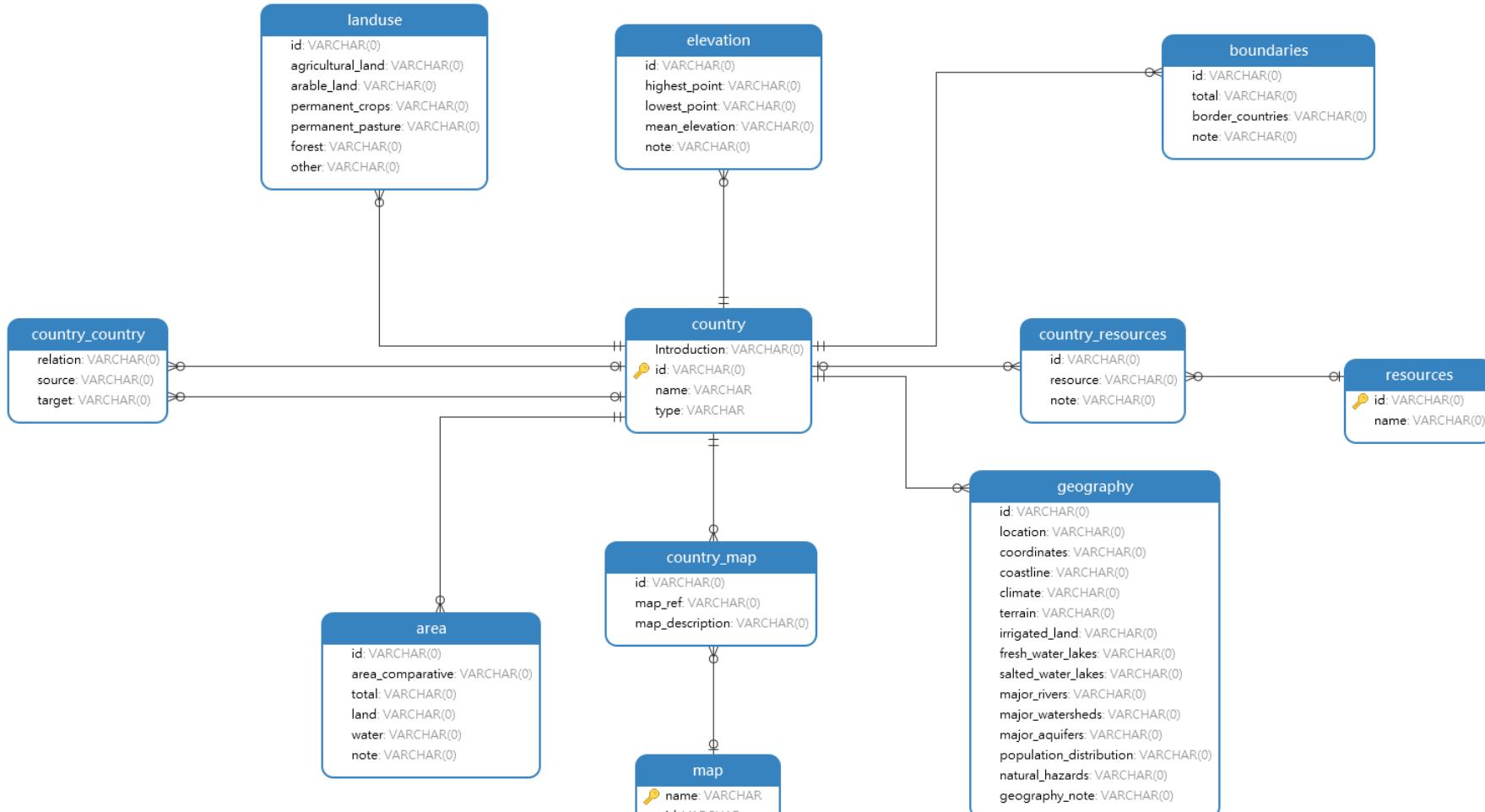
Correction

```
#routes/generales.py
```

```
...
@app.route("/pays")
def pays():
    donnees = []
    for country in Country.query.all():
        donnees.append({
            "nom": country.name,
            "capitale": "inconnu",
            "continent": "inconnu"
        })

    return render_template("pages/tous_pays.html",
        donnees=donnees, sous_titre="Tous les pays")
```

Relations entre classes



Relations one-to-many

```
class A(db.Model):
    cle_primaire = db.Column(db.String(100), primary_key=True)
    un_champ_de_la_table = db.Column(db.String(100))

    propriete_de_relation = db.relationship(
        "classe_liee",
        backref="classes_liees",
        lazy="dynamic"
    )
class classe_liee(db.Model):
    un_champ_de_la_table = db.Column(db.String(100))

    a_id = db.Column(db.String(100), db.ForeignKey('a.cle_primaire'))
```

Exercice

La relation entre les tables `country` et `area` est une relation one-to-many. A partir de la classe `Country()` écrite précédemment dans le cours, ajouter la relation avec la table `area` :

- écrire la classe `Area()`
- écrire la propriété de relation du côté de `Country()`
- ajouter la clé étrangère dans `Area()`

Correction

```
class Country(db.Model):
    __tablename__ = "country"
    id = db.Column(db.String(10), primary_key=True)
    ...
    areas = db.relationship(
        'Area',
        backref='areas',
        lazy=True)

class Area(db.Model):
    __tablename__ = "area"
    ...
    total = db.Column(db.String(100), primary_key=True)
    ...
    id = db.Column(db.String(100), db.ForeignKey('country.id'))
```

Interroger les relations one-to-many

```
# routes/generales.py /pays

for country in Country.query.all():
    ...
    print(country.areas)
return ...
```

Relations many-to-many

```
table_de_relation_a_vers_b = db.Table(  
    "table_de_relation_a_vers_b",  
    db.Column('a_id', db.String(100), db.ForeignKey('a.id'), primary_key=True),  
    db.Column('b_id', db.String(100), db.ForeignKey('b.id'), primary_key=True))  
  
class A(db.Model):  
    id = db.Column(db.String(100), primary_key=True)  
    un_champ_de_la_table = db.Column(db.String(100))  
  
    bs = db.relationship('B',  
        secondary=table_de_relation_a_vers_b,  
        backref="bs")  
  
class B(db.Model):  
    id = db.Column(db.String(100), primary_key=True)  
    un_champ_de_la_table = db.Column(db.String(100))
```

Exercice

Ecrire dans le modèle de données `factbook.py` la relation many-to-many entre `country` et `map`

Correction

```
# table de relation
country_map = db.Table(
    "country_map",
    db.Column('id', db.String(100), db.ForeignKey('country.id'), primary_key=True),
    db.Column('map_ref', db.String(100), db.ForeignKey('map.name'), primary_key=True)
)
```

Correction

```
class Country(db.Model):
    __tablename__ = "country"
    id = db.Column(db.String(10), primary_key=True)
    ...
    maps = db.relationship(
        'Map',
        secondary=country_map,
        backref="maps"
    )
    ...

class Map(db.Model):
    __tablename__ = "map"
    ...
    name = db.Column(db.Text, primary_key=True)
    ...
```

Interroger les relations many-to-many

```
# routes/generales.py /pays

for country in Country.query.all():
    ...
    print(country.maps)
return ...
```

Interroger les relations many-to-many

```
# routes/generales.py /pays

for country in Country.query.all():
    ...
    print(country.maps[0].name)
return ...
```