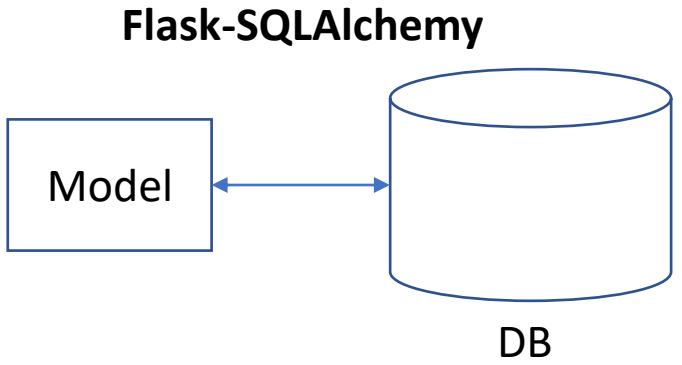
Flask(model)

Flask-SQLAlchemyをインストールして利用します

SQLAlchemy・・・pythonで用いられるORM(ORマッパー)の1つで、SQLを直書きせずにDBを操作するライブラリです。
Flask-SQLAlchemyは、Flaskに合わせて作られたSQLAlchemyです

マイグレーション・・・プログラムのコードからデータベースにテーブルを作成・編集することです



pip install flask-sqlalchemy **# flask-sqlalchemyのインストール**
pip install flask-migrate **# flask-migrateのインストール**

【マイグレーションの手順】
dbの設定を記載しているファイルを設定
export FLASK_APP=myapp.py **# MAC, Linuxの場合(ターミナル)**
set FLASK_APP=myapp.py **# コマンドプロンプトの場合**

flask db init **# migrationsフォルダを作成。マイグレーションに必要なファイルを格納する**
flask db migrate –m “some message” **# テーブルの設定を記載したファイルの内容をmigrationsフォルダに反映する**
flask db upgrade **# migrationsフォルダの内容をDBに登録する**

Flask(DBの設定)

```
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///' + os.path.join(basedir, 'data.sqlite')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app) # SQLAlchemyを扱うためのインスタンス作成
Migrate(app, db) # migrateするDBを設定  DBに接続するための設定
```

(https://flask-sqlalchemy.palletsprojects.com/en/2.x/config/)

SQLALCHEMY_DATABASE_URI	接続するDBのURI **Postgres:** postgresql://username:password@localhost/mydatabase **MySQL:** mysql://username:password@localhost/mydatabase **Oracle**: oracle://username:password@127.0.0.1:1521/sidname **SQLite**: sqlite:////absolute/path/to/foo.db
SQLALCHEMY_TRACK_MODIFICATIONS	これをTrueにすると、Flask-SQLAlchemyがデータベースの変更を追跡管理して、シグナルを発生するようになる。ただし、これを有効にすると追加のメモリが必要となる。
SQLALCHEMY_BINDS	複数のデータベースに接続する際に利用される SQLALCHEMY_BINDS = { 'users': 'mysqldb://localhost/users', 'appmeta': 'sqlite:////path/to/appmeta.db'}

Flask(テーブルの作成)

```
class User(db.Model):
    __tablename__ = 'puppies' # 作成する

    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.Text)
    age = db.Column(db.Integer)

db.create_all() # テーブルを作成


db.session.add_all([sam, john]) # データを挿入

db.session.add(sam) # データを挿入
db.session.add(john) # データを挿入
db.session.commit() # commit
```

DB Browser for SQLiteをインストールする

DBのカラム一覧(https://flask-sqlalchemy.palletsprojects.com/en/2.x/models/)

Integer	数値型。
String(size)	固定文字列
Text	可変文字列
DateTime	タイムスタンプ
Float	浮動小数点数
Boolean	正負値
PickleType	PythonのPickleオブジェクト
LargeBinary	大きいバイナリーデータ

Flask(modelでカラムにオプションを追加)

モデルでテーブルを定義する際に、カラムに制約、インデックスなどを追加することができます。

以下のようにカラム宣言の際にオプションを追加します

オプション	**制約**	**使用例**
primary_key	主キー制約 (ユニーク+NOT NULL + インデックス)	db.Column(db.Integer, primary_key=True)
unique	ユニーク制約 (同じ値を入れられない)	db.Column(db.Integer, unique=True)
nullable	NOT NULL 制約 (NULL値を入れられない)	db.Column(db.Integer, nullable=False)
CheckConstraint	チェック制約 (自由に制約を作成する)	__table_args__ = (CheckConstraint('update_at > create_at'),)
index	インデックスを作成 (索引。検索の際に高速化できる)	db.Column(db.Text, index=True)
db.Index	インデックスを作成	db.Index(“some_index”, func.lower(Person.name)) # 関数インデックス
server_default	デフォルト値の追加	db.Column(db.Text, server_default=A')

Flask(SQLAlchemyの基本操作)

データの挿入
db.session.add() **# 単一のレコードの挿入**
db.session.add_all([]) **# 複数のレコードの挿入**

データの削除
Table.query.delete() **# 単一のレコードの削除**

データの取り出し
Table.query.get(1) **# 主キーで絞り込んで取り出し**
Table.query.all() **# データをすべてリストにして取り出し**
Table.query.first() **# データの最初の要素だけ取り出し**

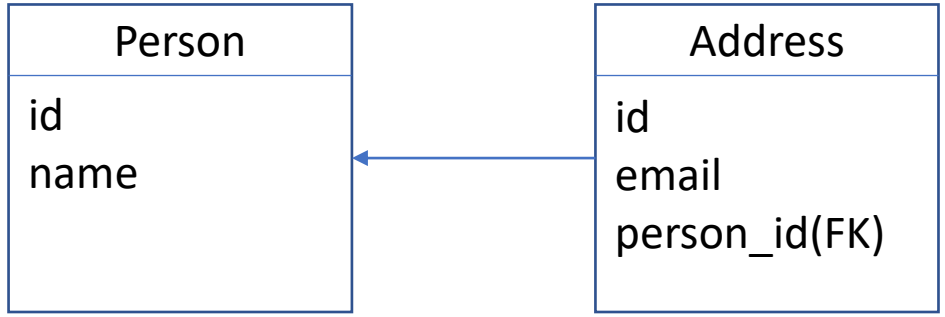
データの絞り込み
Table.query.filter_by(name='A') **# カラムnameがAのデータのみに絞り込み**
Table.query.filter(Table.age > 10) **# カラムageが10より大きいもののみ絞り込み**
Table.query.filter(Table.name.startswith('A')) **# カラムnameがaで始まるもののみ絞り込み**
Table.query.filter(Table.name.endswith('A')) **# カラムnameがaで終わるもののみ絞り込み**
Table.query.limit(1) **# limitで指定した分だけ件数を絞り込む**

データの更新
Table.query.update({'column': 'value'}) **# columnをvalueでupdateする**

Flask(model外部キー)

```
class Person(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    addresses = db.relationship('Address', backref='person', lazy=True)

class Address(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), nullable=False)
    person_id = db.Column(db.Integer, db.ForeignKey('person.id'), nullable=False)
```



db.relationship('Address', backref='person', lazy=True) **# Personテーブルからデータを取得した場合に、Addressも取得するための設定**

backref: Addressからデータを取得した場合のキー
lazy: テーブルを紐づける場合の処理方式
uselist: Falseにすると1対1で紐づける
join_depth: 他のテーブルと紐づける深さを決める

db.ForeignKey('tablename.id')
# 別のTableへの外部キーを作成

lazyのオプション一覧

select	デフォルト。テーブルを紐づける場合にselectを都度実行する
joined	JOIN句でテーブルを紐づける
subquery	サブクエリーでテーブルを紐づける
dynamic	紐づけたテーブルでQuery実行用オブジェクトを生成