INF5190 - MVC - Modèle (suite)

Jean-Philippe Caissy

25 septembre 2019

Types de base de données

Un article qui décrit avec plus de profondeurs les différents typse de base de données :

Comparing Database Types: How Database Types Evolved to Meet Different Needs : https://www.prisma.io/blog/comparison-of-database-models-1iz9u29nwn37

Patron de conception `Active Record`

- Interface qui imite les opérations sur une base de donnée :
  - `SELECT`, `DELETE`, `INSERT`, `UPDATE`
- Le patron représente une approche pour accéder des données à une BD
- Contient habituellement les fonctionnalités suivantes :
  - Récupération d'un objet ou de plusieurs objet
  - Insertion d'une nouvel objet
  - Modification d'un objet existant
  - Suppression d'un objet
- S'occupe de la validation des données
- Assure l'intégralité des contraintes au niveau applicatif (i.e.: Foreign Key)

Patron de conception `Active Record`

Implémentation générale

- Chaque table correspond à une classe
- Chaque colonne d'une table est représentée par des attributs de classe
  - Les attributs de classes sont explicitement liés aux colonnes de la classe
  - Ils définissent le type (`INT`, `VARCHART`, etc) et assurent une validation
- La classe possède des méthodes de classes pour charger des objets

Patron de conception Active Record

Exemple

Voici un exemple d'implémentation :

```
voiture = Voiture()
voiture.marque = "Volkswagen"
voiture.modele = "Golf"
voiture.annee = 2019
voiture.save()
```

L'appel à `voiture.save()` va lancer cette commande SQL :

```
INSERT INTO voitures (marque, modele, annee)
    VALUES ('Volkswaggen', 'Golf', 2019);
```

Patron de conception Active Record

Exemple

L'implémentation de la classe pourrait être utilisé pour récupérer une ligne dans une table de la base de donnée.

```
voiture = Voiture.find_by_id(1)
```

Va rouler la requête SQL sur la base de donnée :

```
SELECT * FROM voitures WHERE id = 12;
```

Et on peut récupérer les informations sur l'objet :

```
print (voiture.id, voiture.marque)
# Affiche : 1 Volkswagen
```

Patron de conception Active Record

Exemple

Le fichier **models.py** de l'exemple d'application Flask implémente une partie du patron de conception Active Record.

```
class Poll(object):
    @classmethod
    def get_poll_by_id(cls, id):
        db = get_db()
        raw_poll = get_db().execute("SELECT id, name,
↪   date FROM polls WHERE id = ?", [id]).fetchone()
        if raw_poll:
            return cls(id=raw_poll[0],
             ↪  name=raw_poll[1], date=raw_poll[2])
        return None

poll = Poll.get_poll_by_id(12)
# Va exécuter :
# SELECT id, name, date FROM polls WHERE id = 12;
```

Object Relation Manager (ORM)

Le patron `Active Record` est généralement utilisé par des outils de persistances et dans les mappings objet-relationnel (Object Relation Manager, *ORM*).

- Dans les langages orienté objet, la gestion de l'accès des données est largement simplifié avec les ORM et le patron Active Record
- Les ORMs exposent les fonctionnalités de bases :
  - Pas besoin de ré-écrire la couche d'abstraction de donnée

Object Relation Manager (ORM)

Un ORM est composé de trois couches :

1. L'abstraction d'accès à la base de donnée (i.e.: SQL)
2. La gestion bi-directionnel des données entre la persistance (BD) et la représentation mémoire (couche du domaine d'affaire)
3. Le patron `Active Record` qui expose en objet les données

La partie 1 (abstraction d'accès aux données) et la partie 3 (`Active Record`) sont abstraits par l'ORM. Il ne reste qu'à définir l'association bi-directionnelle des données

Peewee

Figure 1: Logo de Peewee

- Peewee est un ORM petit et simple pour Python.
- Il s'occupe d'abstraire l'accès à la base de donnée
  - Ce qui veut dire qu'on peut interchanger entre `sqlite`, `postgres` et `mysql`
- Implémente le patron `Active Record`
- Possède un outil de migration de schéma

Peewee

Principes de bases

- **Class héritant `peewee.Model`** : Table de la BD
- **Instance de classe** : Une ligne de la base de donnée
- **Attribut d'instance** : Une colonne de la table

Très similaire à mon implémentation pour la station de vote! Le tout est tout simplement automatisé.

Peewee

Principes de bases

```
from peewee import SqliteDatabase, Model, CharField,
↪  DateField

db = SqliteDatabase('people.db')

class Poll(Model):
    name = CharField()
    date = DateField()

    class Meta:
        database = db
```

N.B. : `peewee` va implicitement rajouter une colonne `id` en tant que clé primaire.

Peewee

Principes de bases

Types d'attributs

`Peewee` permet les associations de types suivants :

Peewee	Type SQL	Description
`AutoField`	`Integer`	Entier auto-incrémentale
`IntegerField`	`Integer`	Entier
`FloatField`	`Real`	Nombre flottant
`CharField`	`Varchar`	String avec taille fixe
`TextField`	`Text`	Texte
`DateField`	`Date`	Date
`DateTimeField`	`Datetime`	Date et heure
`BooleanField`	`Integer` (selon la BD)	Champ booléan
`ForeignKeyField`	`Integer`	Clé étrangère

Et plusieurs autres disponibles dans la **documentation officielle**

Peewee

Principes de bases

Requête sur une table

- La méthode `.get()` sur un modèle permet de récupérer un seul enregistrement d'une table
- Pour récupérer à partir de la clé primaire, la méthode `.get_by_id()` est utilisé
- Lorsqu'un objet n'existe pas, une exception est retournée

```
Poll.get(Poll.id == 1)
Poll.get_by_id(1) # Identique à la méthode précédente
Poll[1] # Aussi identique au précédent

Poll.get(Poll.name == "Mon premier sondage")

Poll.get(Poll.name == "N'existe pas")
# Retourne une exception PollDoesNotExist
```

Peewee

Principes de bases

Requête sur une table

- `.get_or_none()` retourne None au lieu d'une exception

```
Poll.get(Poll.id == 97654)
# None
```

- La méthode `.select()` permet d'itérer sur plusieurs enregistrements

```
Poll.select() # tous lse enregistrements

for poll in Poll.select(Poll.id > 5):
    print(poll.name)
```

Peewee

Principes de bases

Clé étrangère

```
class Poll(Model):
    name = CharField()
    date = DateField()
    # [...]

class Choice(Model):
    poll = ForeignKey(Poll, backref='choices')
    choice = CharField()
    # [...]
```

Peewee utilise le type `ForeignKey` pour définir une contrainte d'association avec clé étrangère. Chaque champ étranger a une relation inverse implicite.

Peewee

Principes de bases

Clé étrangère

```
from datetime import datetime

Poll.create(name="Mon premier sondage",
 ↪  date=datetime.now())
Choice.create(choice="Premier choix", poll=poll)
Choice.create(choice="Deuxième choix", poll=poll)

poll= Poll.first()
poll.choices[0].name # Premier choix
poll.choices[1].name # Deuxième choix
```

Peewee

Principes de bases

Contraintes

On peut définir des contraintes d'unicité d'intégrité et d'index sur les champs:

```
from peewee import Model, CharField, DecimalField,
↪  DateTimeField, Check, SQL


class Product(Model):
    name = CharField(unique=True)
    code = CharField(index=True)
    price = DecimalField(
        constraints=[Check('price < 10000')]
    )
    created = DateTimeField(
        constraints=[
            SQL("DEFAULT (datetime('now'))")
    ])
```

Peewee

Création du schéma

- Une fois que le modèle initiale est définit, `Peewee` peut créer les tables définies.

```
db = SqliteDatabase('./database.sqlite')

db.connect()

db.create_tables([Poll, Choice])
```

Peewee

La documentation contient beaucoup plus de détails sur l'API de Peewee :

http://docs.peewee-orm.com/en/latest/index.html

Liens

- The Active Record and Data Mappers of ORM Pattern
- Documentation de Peewee