# Définir et accéder à la base de données

L'application utilisera une base de données SQLite pour stocker les utilisateurs et les messages. Python offre un support intégré pour SQLite dans le module **sqlite3**.

SQLite est pratique car il ne nécessite pas la mise en place d'un serveur de base de données distinct et est intégré à Python. Cependant, si des requêtes SQL concurrentes essaient d'écrire en même temps dans la base de données, elles ralentiront car chaque écriture se fait de manière séquentielle. Les petites applications ne le remarqueront pas. Une fois que vous aurez atteint une certaine taille, vous voudrez peut-être passer à une autre base de données.

Le tutoriel n'entre pas dans les détails de SQL. Si vous n'êtes pas familier avec ce langage, les documents de SQLite décrivent le language.

## Connectez-vous à la base de données

La première chose à faire lorsqu'on travaille avec une base de données SQLite (et la plupart des autres bibliothèques de bases de données Python) est de créer une connexion à celle-ci. Toutes les requêtes et opérations SQL sont effectuées en utilisant cette connexion, qui est fermée une fois le travail terminé.

Dans les applications Web, cette connexion est généralement liée à la requête HTTP en cours. Elle est créée à un moment donné lors du traitement d'une requête HTTP, et fermée avant l'envoi de la réponse.

`flaskr/db.py`

```
import sqlite3

import click
from flask import current_app, g
from flask.cli import with_appcontext


def get_db():
    if 'db' not in g:
        g.db = sqlite3.connect(
            current_app.config['DATABASE'],
            detect_types=sqlite3.PARSE_DECLTYPES
        )
        g.db.row_factory = sqlite3.Row

    return g.db
```

```
def close_db(e=None):
    db = g.pop('db', None)

    if db is not None:
        db.close()
```

**g** est un objet spécial qui est unique pour chaque requête HTTP. Il est utilisé pour stocker les données qui pourraient être accédées par plusieurs fonctions au cours de la requête. La connexion est stockée et réutilisée au lieu de créer une nouvelle connexion si `get_db` est appelé une seconde fois dans la même requête.

**current_app** est un autre objet spécial qui pointe vers l'application Flask qui traite la requête HTTP. Puisque vous avez utilisé une fabrique d'application, il n'y a pas d'objet application lorsque vous écrivez le reste de votre code. `get_db` sera appelé lorsque l'application aura été créée et traitera une requête, donc **current_app** peut être utilisé.

**sqlite3.connect()** établit une connexion au fichier pointé par la clé de configuration `DATABASE`. Ce fichier n'a pas besoin d'exister encore, et n'existera pas tant que vous n'aurez pas initialisé la base de données plus tard.

**sqlite3.Row** indique à la connexion de retourner des lignes qui se comportent comme des dictionnaires. Cela permet d'accéder aux colonnes par leur nom.

`close_db` vérifie si une connexion a été créée en vérifiant si `g.db` a été défini. Si la connexion existe, elle est fermée. Plus loin, vous indiquerez à votre application la fonction `close_db` dans la fabrique de l'application afin qu'elle soit appelée après chaque requête HTTP.

# Créer les tables

Dans SQLite, les données sont stockées dans des *tables* et des *colonnes*. Ceux-ci doivent être créés avant que vous puissiez stocker et récupérer des données. Flaskr va stocker les utilisateurs dans la table `user`, et les messages dans la table `post`. Créez un fichier avec les commandes SQL nécessaires pour créer des tables vides :

flaskr/schema.sql

```
DROP TABLE IF EXISTS user;
DROP TABLE IF EXISTS post;

CREATE TABLE user (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT UNIQUE NOT NULL,
  password TEXT NOT NULL
);

CREATE TABLE post (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  author_id INTEGER NOT NULL,
```

```
  created TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  title TEXT NOT NULL,
  body TEXT NOT NULL,
  FOREIGN KEY (author_id) REFERENCES user (id)
);
```

Ajoutez les fonctions Python qui exécuteront ces commandes SQL au fichier `db.py` :

flaskr/db.py

```
def init_db():
    db = get_db()

    with current_app.open_resource('schema.sql') as f:
        db.executescript(f.read().decode('utf8'))


@click.command('init-db')
@with_appcontext
def init_db_command():
    """Clear the existing data and create new tables."""
    init_db()
    click.echo('Initialized the database.')
```

**`open_resource()`** ouvre un fichier relatif au paquet `flaskr`, ce qui est utile puisque vous ne saurez pas nécessairement où se trouve cet emplacement lors du déploiement ultérieur de l'application. `get_db` retourne une connexion à la base de données, qui est utilisée pour exécuter les commandes lues dans le fichier.

**`click.command()`** définit une commande de ligne de commande appelée `init-db` qui appelle la fonction `init_db` et affiche un message de réussite à l'utilisateur. Vous pouvez lire Command Line Interface pour en savoir plus sur l'écriture des commandes.

# S'enregistrer auprès de l'application

Les fonctions `close_db` et `init_db_command` doivent être enregistrées avec l'instance de l'application ; sinon, elles ne seront pas utilisées par l'application. Cependant, puisque vous utilisez une fonction de fabrique, cette instance n'est pas disponible lors de l'écriture des fonctions. Au lieu de cela, écrivez une fonction qui prend une application et effectue l'enregistrement.

flaskr/db.py

```
def init_app(app):
    app.teardown_appcontext(close_db)
    app.cli.add_command(init_db_command)
```

**app.teardown_appcontext()** indique à Flask d'appeler cette fonction lors du nettoyage après le renvoie de la réponse.

**app.cli.add_command()** ajoute une nouvelle commande qui peut être appelée avec la commande `flask`.

Importez et appelez cette fonction depuis la fabrique. Placez le nouveau code à la fin de la fonction de la fabrique avant de retourner l'application.

flaskr/__init__.py

```
def create_app():
    app = ...
    # existing code omitted

    from . import db
    db.init_app(app)

    return app
```

# Initialiser le fichier de la base de données

Maintenant que `init-db` a été enregistré avec l'application, il peut être appelé en utilisant la commande `flask`, similaire à la commande `run` de la page précédente.

---

## Note:

Si vous utilisez toujours le serveur de la page précédente, vous pouvez soit arrêter le serveur, soit exécuter cette commande dans un nouveau terminal. Si vous utilisez un nouveau terminal, n'oubliez pas de vous rendre dans le répertoire de votre projet et d'activer l'environnement virtuel comme décrit dans l'Installation. Vous devrez également définir `FLASK_APP` et `FLASK_ENV` comme indiqué sur la page précédente.

---

Exécutez la commande `init-db` :

```
$ flask init-db
Initialized the database.
```

Il y aura maintenant un fichier `flaskr.sqlite` dans le dossier `instance` de votre projet.

Continuer vers Blueprints et vues.