

## **# Pour construire ces tables dans le fichier development.db, vous pouvez suivre les étapes ci-dessous :**

---

### **## Étape 1 : Préparer l'environnement**

1. **Assurez-vous que SQLite est installé** : Vérifiez avec :

```
```bash
sqlite3 --version
```
```

Si ce n'est pas le cas, installez SQLite (sous Ubuntu, par exemple) :

```
```bash
sudo apt install sqlite3
```
```

2. **Créez ou ouvrez le fichier development.db** :

```
```bash
sqlite3 instance/development.db
```
```

3. **Activez les clés étrangères** (important pour gérer les relations) :

```
```sql
PRAGMA foreign_keys = ON;
```
```

---

### **## Étape 2 : Coller et exécuter les commandes SQL**

Une fois dans l'interface SQLite, collez vos commandes SQL pour créer les tables.

```
```sql
-- Create User table
CREATE TABLE users (
    id CHAR(36) PRIMARY KEY,
    first_name VARCHAR(255),

```

```

last_name VARCHAR(255),

email VARCHAR(255) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

is_admin BOOLEAN DEFAULT FALSE

);

-- Create Place table

CREATE TABLE places (

    id CHAR(36) PRIMARY KEY,

    title VARCHAR(255) NOT NULL,

    description TEXT,

    price DECIMAL(10, 2) NOT NULL,

    latitude FLOAT NOT NULL,

    longitude FLOAT NOT NULL,

    owner_id CHAR(36),

    FOREIGN KEY (owner_id) REFERENCES users(id) ON DELETE CASCADE

);

-- Create Review table

CREATE TABLE reviews (

    id CHAR(36) PRIMARY KEY,

    text TEXT NOT NULL,

    rating INT CHECK (rating BETWEEN 1 AND 5),

    user_id CHAR(36),

    place_id CHAR(36),

    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,

    FOREIGN KEY (place_id) REFERENCES places(id) ON DELETE CASCADE,

    UNIQUE (user_id, place_id)

```

```
);
```

```
-- Create Amenity table
```

```
CREATE TABLE amenities (  
    id CHAR(36) PRIMARY KEY,  
    name VARCHAR(255) UNIQUE NOT NULL  
);
```

```
-- Create Place_Amenity association table
```

```
CREATE TABLE place_amenity (  
    place_id CHAR(36),  
    amenity_id CHAR(36),  
    PRIMARY KEY (place_id, amenity_id),  
    FOREIGN KEY (place_id) REFERENCES places(id) ON DELETE CASCADE,  
    FOREIGN KEY (amenity_id) REFERENCES amenities(id) ON DELETE CASCADE  
);
```

```
...
```

---

### ## Étape 3 : Vérifiez que les tables ont été créées

1. **Lister les tables créées** : Une fois les commandes exécutées, utilisez :

```
```sql  
  
.tables  
  
```
```

Vous devriez voir les tables suivantes :

```
```bash  
  
amenities    place_amenity places    reviews    users  
  
```
```

2. **Vérifiez la structure d'une table** (par exemple users) :

```
```sql
```

```
PRAGMA table_info(users);
```

```
```
```

---

## **## Étape 4 : Insérer des données de test**

Ajoutez des données pour vérifier que les relations fonctionnent correctement.

### **1. Ajouter un utilisateur :**

```
```sql
```

```
INSERT INTO users (id, first_name, last_name, email, password, is_admin)
```

```
VALUES ('user-uuid-1', 'John', 'Doe', 'john.doe@example.com', 'hashed_password', FALSE);
```

```
```
```

### **2. Ajouter un lieu associé à l'utilisateur :**

```
```sql
```

```
INSERT INTO places (id, title, description, price, latitude, longitude, owner_id)
```

```
VALUES ('place-uuid-1', 'Lovely Apartment', 'A cozy apartment in the city center.', 150.00,  
48.8566, 2.3522, 'user-uuid-1');
```

```
```
```

### **3. Ajouter un avis lié à l'utilisateur et au lieu :**

```
```sql
```

```
INSERT INTO reviews (id, text, rating, user_id, place_id)
```

```
VALUES ('review-uuid-1', 'Fantastic place!', 5, 'user-uuid-1', 'place-uuid-1');
```

```
```
```

### **4. Ajouter une commodité :**

```
```sql
```

```
INSERT INTO amenities (id, name)
```

```
VALUES ('amenity-uuid-1', 'WiFi');
```

```
```
```

### **5. Associer une commodité au lieu :**

```
```sql
```

```
INSERT INTO place_amenity (place_id, amenity_id)
```

```
VALUES ('place-uuid-1', 'amenity-uuid-1');
```

```
...
```

---

### ## Étape 5 : Vérifiez les relations

#### 1. Afficher tous les lieux d'un utilisateur :

```
```sql
```

```
SELECT * FROM places WHERE owner_id = 'user-uuid-1';
```

```
...
```

#### 2. Afficher tous les avis d'un lieu :

```
```sql
```

```
SELECT * FROM reviews WHERE place_id = 'place-uuid-1';
```

```
...
```

#### 3. Afficher les commodités associées à un lieu :

```
```sql
```

```
SELECT a.name
```

```
FROM amenities a
```

```
JOIN place_amenity pa ON a.id = pa.amenity_id
```

```
WHERE pa.place_id = 'place-uuid-1';
```

```
...
```

---

### ## Étape 6 : Automatiser avec un fichier SQL

#### 1. Créer un fichier `setup.sql` contenant toutes les commandes SQL :

- Placez toutes les commandes CREATE TABLE et les données de test dans le fichier.

#### 2. Exécuter le script SQL automatiquement :

```
```bash
```

```
sqlite3 instance/development.db < setup.sql
```

```
...
```

---

## ## Étape 7 : Intégration avec Flask

Dans votre application Flask :

1. Configurez `SQLALCHEMY_DATABASE_URI` dans `config.py` pour pointer vers `instance/development.db` :

```
```python
```

```
SQLALCHEMY_DATABASE_URI = "sqlite:///instance/development.db"
```

```
SQLALCHEMY_TRACK_MODIFICATIONS = False
```

```
```
```

2. Utilisez SQLAlchemy pour interagir avec ces tables dans votre application.

---

Avec ces étapes, vos tables seront correctement créées dans `development.db` et prêtes à être utilisées. Si vous rencontrez des problèmes, partagez les messages d'erreur ou la sortie pour une assistance plus approfondie !

## 1. Base de données `development.db` :

- Fichier SQLite automatiquement généré dans `instance/`.
- Migrations Alembic remplacées par une initialisation manuelle pour coller à la consigne.
- **Commandes de création :**

```
bash
Copier le code
flask shell
>>> from app import db
>>> db.create_all()
```




- **Tables disponibles :**

- `users`
- `places`
- `reviews`
- `amenities`
- `place_amenity` (table association Many-to-Many).

## 2. Tests sur la base de données :

- **CRUD sur chaque table vérifié avec succès :**
  - Insertion d'un `admin user` en tant que superutilisateur.
  - Ajout d'une place avec une relation valide à un utilisateur.
  - Test de suppression en cascade sur les dépendances.

## 3. Tests des endpoints :

- **POST `/api/v1/users` :**  Création d'utilisateur avec email unique.
- **POST `/api/v1/places` :**  Ajout avec géolocalisation (latitude/longitude).
- **GET `/api/v1/reviews` :**  Filtrage et récupération des commentaires liés à un lieu.

## 4. Documentation enrichie :

- **Ajout de commandes SQL dans `README.md` pour initialiser la base :**

```
plaintext
Copier le code
sqlite3 instance/development.db
PRAGMA foreign_keys=ON;
```

- Guide pour vérifier les relations SQL avec des `JOIN`.