Flask with Databases

1. Setting Up Flask with SQLAlchemy

Here's a step-by-step guide to integrating Flask with SQLAlchemy:

Step 1: Install Flask and SQLAlchemy

First, you need to install Flask and SQLAlchemy. You can do this using `pip`:

```
pip install Flask
pip install Flask-SQLAlchemy
```

Step 2: Set Up Flask Application

Create a basic Flask application and configure it to use SQLAlchemy.

python

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# Configuring the database URI, example using SQLite
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

# Initializing the SQLAlchemy object
db = SQLAlchemy(app)

# Define your models (tables)
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
```

```
    email = db.Column(db.String(120), unique=True, nullable=False)

    def __repr__(self):
        return f'<User {self.username}>'

# Creating the database tables
with app.app_context():
    db.create_all()

if __name__ == '__main__':
    app.run(debug=True)
```

Step 3: Defining Models

In the example above, `User` is a model that represents a table in the database. Each model class inherits from `db.Model`, and each attribute represents a column in the table.

- `id` is an auto-incrementing primary key.
- `username` and `email` are string columns, and `unique=True` ensures no duplicate entries for these columns.
- `nullable=False` means these columns cannot be empty.

Step 4: Creating the Database

The `db.create_all()` command creates all tables defined in the models. This command should be run within the application context, hence it's placed inside `with app.app_context():` block.

Step 5: Working with the Database

You can now interact with the database using SQLAlchemy's ORM capabilities.

Adding Data:

python

```
new_user = User(username='alice', email='alice@example.com')
db.session.add(new_user)
db.session.commit()
```

Querying Data:

python

```
# Query all users
users = User.query.all()

# Query by username
alice = User.query.filter_by(username='alice').first()

print(users)
```

Updating Data:

python

```
alice = User.query.filter_by(username='alice').first()
alice.email = 'alice_new@example.com'
db.session.commit()
```

Deleting Data:

python

```
alice = User.query.filter_by(username='alice').first()
db.session.delete(alice)
db.session.commit()
```

2. Using Other ORM Frameworks

While SQLAlchemy is the most common ORM used with Flask, there are other ORMs you can use depending on your needs.

Peewee

Peewee is a small and simple ORM. It's more lightweight than SQLAlchemy, making it a good choice for smaller projects.

Installation:

```
pip install peewee
```

-

Integration: Peewee can be integrated into a Flask app, but you need to manually manage the connection and models.
Example of defining models with Peewee:
python

```
from peewee import *

db = SqliteDatabase('example.db')

class BaseModel(Model):
    class Meta:
        database = db

class User(BaseModel):
    username = CharField(unique=True)
    email = CharField(unique=True)

# Create the tables
db.connect()
db.create_tables([User])
```

-

Django ORM with Flask

Though not common, you can use Django ORM with Flask. This is more complex and generally not recommended unless you're already committed to using Django ORM in your project.

3. Database Migrations

When your database schema changes, it's crucial to manage these changes effectively. Flask-Migrate, which is based on Alembic, helps with this.

Install Flask-Migrate:

```
pip install Flask-Migrate
```

-

Set Up Flask-Migrate:
python

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///example.db'
db = SQLAlchemy(app)
migrate = Migrate(app, db)
```

-

Initialize Migration Repository:

```
flask db init
```

-

Create Migrations:

```
flask db migrate -m "Initial migration."
```

-

Apply Migrations:

```
flask db upgrade
```

-

4. Working with Other Databases

SQLAlchemy supports various databases, including MySQL, PostgreSQL, and SQLite. You just need to change the `SQLALCHEMY_DATABASE_URI` accordingly.

- **SQLite:** `'sqlite:///example.db'`
- **PostgreSQL:** `'postgresql://username:password@localhost/dbname'`
- **MySQL:** `'mysql+pymysql://username:password@localhost/dbname'`

Summary

- **SQLAlchemy** is the go-to ORM for Flask, providing a robust and Pythonic way to interact with relational databases.
- **Peewee** is a lightweight alternative for smaller projects.
- **Database Migrations** can be managed using Flask-Migrate, which helps track changes in your database schema.