

## LAB 7:

### Database Management with Flask-SQLAlchemy and Email Support with Flask-Mail

#### Nội dung

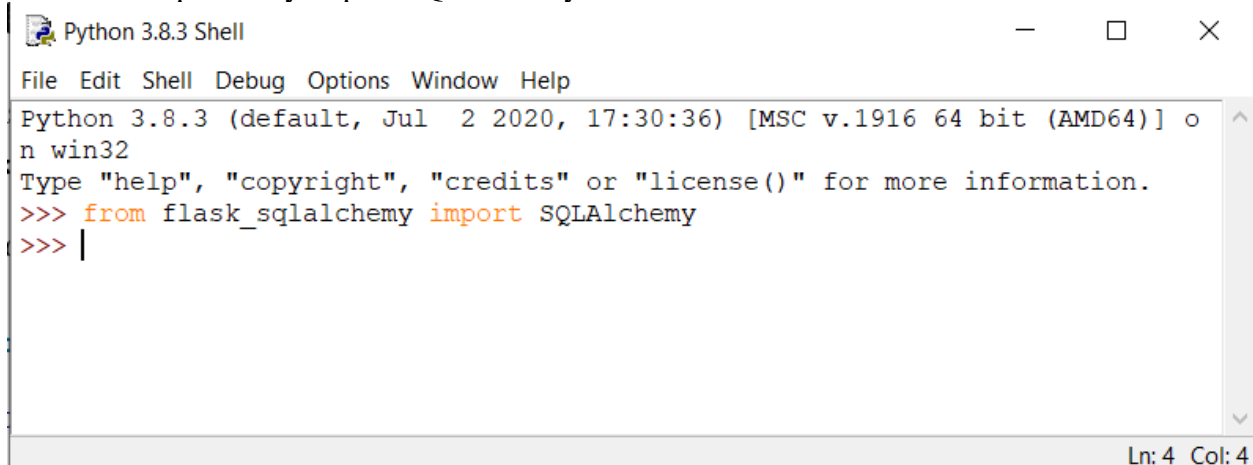
- Kiểm tra cài đặt gói Flask-SQLAlchemy trên máy (nếu cần)
- Cấu hình cơ sở dữ liệu
- Tạo bảng dữ liệu từ Flask-SQLAlchemy
- Tạo các ràng buộc
- Thao tác với bảng
- Truy vấn một đối tượng trong bảng
- Các mối quan hệ cơ sở dữ liệu động
- Tích hợp với Python Shell
- Tạo Migration Repository
- Email Support with Flask-Mail
- Gửi Email bằng Python Shell
- Tích hợp Email với Ứng dụng
- Tùy chọn cấu hình
- Tập lệnh ứng dụng
- Kiểm thử đơn vị

#### Bài 1: Kiểm tra cài đặt gói Flask-SQLAlchemy trên máy (nếu cần)

##### Thực hiện:

Trong IDLE, chúng ta dễ dàng kiểm tra thông qua lệnh

```
from flask_sqlalchemy import SQLAlchemy
```



The screenshot shows a Python 3.8.3 Shell window with the following content:

```
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> from flask_sqlalchemy import SQLAlchemy
>>> |
```

The status bar at the bottom right indicates "Ln: 4 Col: 4".

Ngoài ra, chúng ta có thể sử dụng lệnh: **pip install --pre SQLAlchemy** để kiểm tra các thông tin của gói Flask-SQLAlchemy trên hệ thống khi đã cài đặt.

```

Select Anaconda Prompt (anaconda3)

no such option: -p

(base) C:\Users\nguye>pip install --pre SQLAlchemy
Requirement already satisfied: SQLAlchemy in c:\users\nguye\anaconda3\lib\site-packages (1.3.18)

(base) C:\Users\nguye>
(base) C:\Users\nguye>
(base) C:\Users\nguye>

```

## Bài 2: Cấu hình cơ sở dữ liệu:

Sinh viên sử dụng IDLE có thể tạo mới 1 tập tin đặt tên là hello.py như sau:

*Example 5-1. hello.py: database configuration*

```

import os
from flask_sqlalchemy import SQLAlchemy

basedir = os.path.abspath(os.path.dirname(__file__))
app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] = \
    'sqlite:/// ' + os.path.join(basedir, 'data.sqlite')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

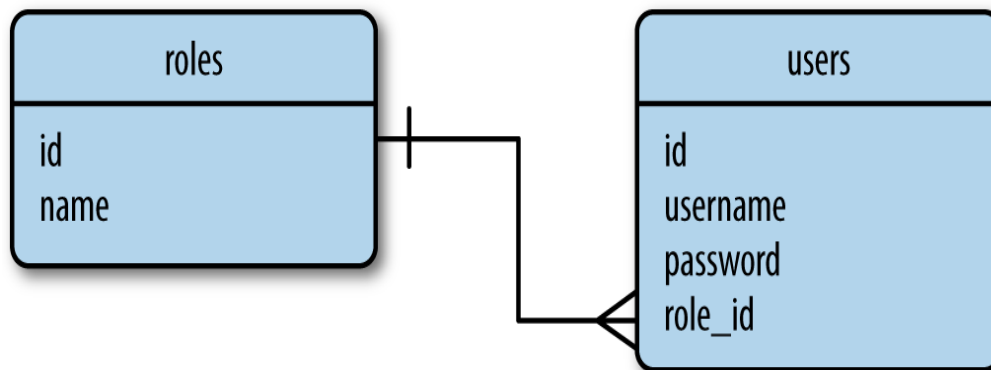
db = SQLAlchemy(app)

```

Sau khi tạo xong, sinh viên bấm F5 để thực thi chương trình

## Bài 3: Tạo bảng dữ liệu từ Flask-SQLAlchemy

Sinh viên tạo các bảng dữ liệu và ràng buộc dữ liệu như hình sau:



**Thực hiện:**

*Example 5-2. hello.py: Role and User model definition*

```
class Role(db.Model):
    __tablename__ = 'roles'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(64), unique=True)

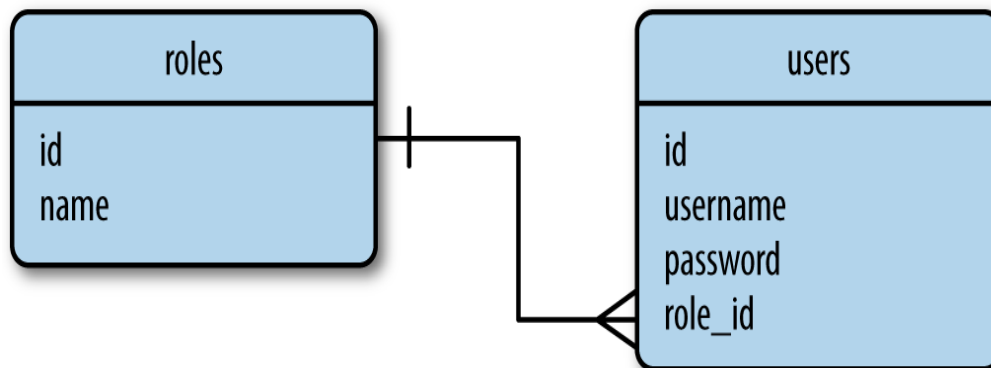
    def __repr__(self):
        return '<Role %r>' % self.name

class User(db.Model):
    __tablename__ = 'users'
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), unique=True, index=True)

    def __repr__(self):
        return '<User %r>' % self.username
```

#### Bài 4: Tạo các ràng buộc

Sinh viên tạo ràng buộc theo hình sau



Thực hiện:

*Example 5-3. hello.py: relationships in the database models*

```
class Role(db.Model):
    # ...
    users = db.relationship('User', backref='role')

class User(db.Model):
    # ...
    role_id = db.Column(db.Integer, db.ForeignKey('roles.id'))
```

### Bài 5: Thao tác với bảng

Yêu cầu:

- Yêu cầu 1: Tạo bảng và thêm dữ liệu vào trong bảng (Lưu ý trong khi tạo bảng, hãy xóa những bảng/database đã tạo cùng tên trước đó)
- Yêu cầu 2: Thực hiện các thao tác thêm, sửa, thay đổi trong bảng (bằng 2 cách)

Thực hiện:

Yêu cầu 1: Tạo bảng như sau

```
(venv) $ flask shell
>>> from hello import db
>>> db.create_all()
```

Xóa những bảng/CSDL đã tạo trước đó

```
>>> db.drop_all()
>>> db.create_all()
```

Thêm dữ liệu vào bảng:

```
>>> from hello import Role, User
>>> admin_role = Role(name='Admin')
>>> mod_role = Role(name='Moderator')
>>> user_role = Role(name='User')
>>> user_john = User(username='john', role=admin_role)
>>> user_susan = User(username='susan', role=user_role)
>>> user_david = User(username='david', role=user_role)
```

```
>>> from hello import Role, User
>>> admin_role = Role(name='Admin')
>>> mod_role = Role(name='Moderator')
>>> user_role = Role(name='User')
>>> user_john = User(username='john', role=admin_role)
>>> user_susan = User(username='susan', role=user_role)
>>> user_david = User(username='david', role=user_role)
```

## Yêu cầu 2: Thực hiện các thay đổi

### Cách 1:

```
>>> db.session.add(admin_role)
>>> db.session.add(mod_role)
>>> db.session.add(user_role)
>>> db.session.add(user_john)
>>> db.session.add(user_susan)
>>> db.session.add(user_david)
```

### Cách 2:

```
>>> db.session.add_all([admin_role, mod_role, user_role,
... user_john, user_susan, user_david])
```

To write the objects to the database, the session needs to be *committed* by calling its `commit()` method:

```
>>> db.session.commit()
```

Check the `id` attributes again after having the data committed to see that they are now set:

```
>>> print(admin_role.id)
1
>>> print(mod_role.id)
2
>>> print(user_role.id)
3
```

### Thay đổi 'Admin' thành 'Administrator':

```
>>> admin_role.name = 'Administrator'
>>> db.session.add(admin_role)
>>> db.session.commit()
```

### Xóa một hàng:

```
>>> db.session.delete(mod_role)
>>> db.session.commit()
```

#### Bài 6: Truy vấn một đối tượng trong bảng

```
>>> Role.query.all()
[<Role 'Administrator'>, <Role 'User'>]
>>> User.query.all()
[<User 'john'>, <User 'susan'>, <User 'david'>]
```

Cách khác cụ thể hơn sử dụng bộ lọc:

```
>>> User.query.filter_by(role=user_role).all()
[<User 'susan'>, <User 'david'>]
```

Bài tập: các bạn kiểm tra truy vấn SQL gốc mà SQLAlchemy tạo cho một truy vấn nhất định bằng cách chuyển đổi đối tượng truy vấn thành một chuỗi

#### Bài 7: Các mối quan hệ cơ sở dữ liệu động

```
class Role(db.Model):
    # ...
    users = db.relationship('User', backref='role', lazy='dynamic')
    # ...
```

With the relationship configured in this way, `user_role.users` returns a query that hasn't executed yet, so filters can be added to it:

```
>>> user_role.users.order_by(User.username).all()
[<User 'david'>, <User 'susan'>]
>>> user_role.users.count()
2
```

Sử dụng cơ sở dữ liệu trong các chức năng xem:

```
@app.route('/', methods=['GET', 'POST'])
def index():
    form = NameForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.name.data).first()
        if user is None:
            user = User(username=form.name.data)
            db.session.add(user) db.session.commit()
            session['known'] = False
        else:
            session['known'] = True session['name'] =
            form.name.data form.name.data = ''
        return redirect(url_for('index'))
    return render_template('index.html', form=form,
        name=session.get('name'),
        known=session.get('known', False))
```

## Bài 8: Tích hợp với Python Shell

Thêm một shell context:

```
@app.shell_context_processor
def make_shell_context():
    return dict(db=db, User=User, Role=Role)
```

```
$ flask shell
>>> app
<Flask 'hello'>
>>> db
<SQLAlchemy engine='sqlite:////home/flask/flasky/data.sqlite'>
>>> User
<class 'hello.User'>
```

## Bài 9: Tạo Migration Repository

Cài đặt môi trường Flask-Migrate

Bằng câu lệnh: \$ **pip install flask-migrate**

**Khởi tạo Flask-Migrate**

```
from flask_migrate import Migrate

# ...

migrate = Migrate(app, db)
```

Khi bạn làm việc trên một dự án mới, bạn có thể thêm hỗ trợ cho việc di chuyển cơ sở dữ liệu bằng lệnh con init:

```
(venv) $ flask db init
Creating directory /home/flask/flasky/migrations...done
Creating directory /home/flask/flasky/migrations/versions...done Generating
/home/flask/flasky/migrations/alembic.ini...done Generating
/home/flask/flasky/migrations/env.py...done Generating
/home/flask/flasky/migrations/env.pyc...done Generating
/home/flask/flasky/migrations/README...done Generating
/home/flask/flasky/migrations/script.py.mako...done Please edit
configuration/connection/logging settings in
'/home/flask/flasky/migrations/alembic.ini' before proceeding.
```

Sau khi tập lệnh di chuyển đã được xem xét và chấp nhận, nó có thể được áp dụng cho cơ sở dữ liệu bằng cách sử dụng lệnh nâng cấp db:

```
(venv) $ flask db upgrade

INFO      [alembic.migration] Context impl SQLiteImpl.
INFO      [alembic.migration] Will assume non-transactional DDL.
INFO      [alembic.migration] Running upgrade None -> 1bc594146bb5, initial migration
```

**Bài 10: Email Support with Flask-Mail**

Tải gói Flask-mail bằng câu lệnh: `$ pip install flask-mail`

**Cấu hình Flask-Mail cho Gmail:**

```
import os
# ...
app.config['MAIL_SERVER'] = 'smtp.googlemail.com'
app.config['MAIL_PORT'] = 587 app.config['MAIL_USE_TLS'] =
True
app.config['MAIL_USERNAME'] = os.environ.get('MAIL_USERNAME')
app.config['MAIL_PASSWORD'] = os.environ.get('MAIL_PASSWORD')
```

Khởi tạo Flask-Mail bằng dòng lệnh: `from flask_mail import Mail mail = Mail(app)`



Hai biến môi trường giữ tên người dùng và mật khẩu của máy chủ email cần được xác định trong môi trường. Có thể cài đặt các biến này như sau:

```
(venv) $ set MAIL_USERNAME=<Gmail username>
```

```
(venv) $ set MAIL_PASSWORD=<Gmail password>
```

### Bài 11: Gửi Email bằng Python Shell

Để kiểm tra cấu hình, bạn có thể bắt đầu phiên trình bao và gửi email kiểm tra

```
(venv) $ flask shell
>>> from flask_mail import Message
>>> from hello import mail
>>> msg = Message('test email', sender='you@example.com',
... recipients=['you@example.com'])
>>> msg.body = 'This is the plain text body'
>>> msg.html = 'This is the <b>HTML</b> body'
>>> with app.app_context():
...     mail.send(msg)
...
```

### Bài 12: Tích hợp Email với Ứng dụng

Hỗ trợ email:

```
from flask_mail import Message

app.config['FLASKY_MAIL_SUBJECT_PREFIX'] = '[Flasky]'
app.config['FLASKY_MAIL_SENDER'] = 'Flasky Admin
<flasky@example.com>'

def send_email(to, subject, template, **kwargs):
    msg = Message(app.config['FLASKY_MAIL_SUBJECT_PREFIX'] +
                  subject, sender=app.config['FLASKY_MAIL_SENDER'],
                  recipients=[to])
    msg.body = render_template(template + '.txt', **kwargs)
    msg.html = render_template(template + '.html', **kwargs)
    mail.send(msg)
```

Hỗ trợ một email không đồng bộ:

```
from threading import Thread

def send_async_email(app, msg): with
    app.app_context(): mail.send(msg)

def send_email(to, subject, template, **kwargs):
    msg = Message(app.config['FLASKY_MAIL_SUBJECT_PREFIX'] + subject,
                  sender=app.config['FLASKY_MAIL_SENDER'], recipients=[to])
    msg.body = render_template(template + '.txt', **kwargs) msg.html =
    render_template(template + '.html', **kwargs) thr =
    Thread(target=send_async_email, args=[app, msg]) thr.start()
    return thr
```

Bài 13: Tùy chọn cấu hình

Cấu hình ứng dụng:

```
import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config:
    SECRET_KEY = os.environ.get('SECRET_KEY') or 'hard to guess
    string' MAIL_SERVER = os.environ.get('MAIL_SERVER',
    'smtp.googlemail.com') MAIL_PORT =
    int(os.environ.get('MAIL_PORT', '587'))
    MAIL_USE_TLS = os.environ.get('MAIL_USE_TLS', 'true').lower() in
    \ ['true', 'on', '1']
    MAIL_USERNAME = os.environ.get('MAIL_USERNAME')
    MAIL_PASSWORD = os.environ.get('MAIL_PASSWORD')
    FLASKY_MAIL_SUBJECT_PREFIX = '[Flasky]'
    FLASKY_MAIL_SENDER = 'Flasky Admin <flasky@example.com>'
    FLASKY_ADMIN = os.environ.get('FLASKY_ADMIN')
    SQLALCHEMY_TRACK_MODIFICATIONS = False
```

```

@staticmethod
def init_app(app):
    pass

class DevelopmentConfig(Config):
    DEBUG = True
    SQLALCHEMY_DATABASE_URI = os.environ.get('DEV_DATABASE_URL') or \
        'sqlite:/// ' + os.path.join(basedir, 'data-dev.sqlite')

class TestingConfig(Config):
    TESTING = True
    SQLALCHEMY_DATABASE_URI = os.environ.get('TEST_DATABASE_URL') or \
        'sqlite://'

class ProductionConfig(Config):
    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:/// ' + os.path.join(basedir, 'data.sqlite')

config = {
    'development': DevelopmentConfig, 'testing':
    TestingConfig, 'production': ProductionConfig,

    'default': DevelopmentConfig
}

```

Hàm tạo gói ứng dụng:

```
from flask import Flask, render_template
from flask_bootstrap import Bootstrap
from flask_mail import Mail
from flask_moment import Moment
from flask_sqlalchemy import SQLAlchemy
from config import config

bootstrap = Bootstrap()
mail = Mail()
moment = Moment()
db = SQLAlchemy()

def create_app(config_name):
    app = Flask(__name__)
    app.config.from_object(config[config_name])
    config[config_name].init_app(app)

    bootstrap.init_app(app)
    mail.init_app(app)
    moment.init_app(app)
    db.init_app(app)

    # attach routes and custom error pages here

    return app
```

Bài 14: Tập lệnh ứng dụng

Main Script

```
import os
from app import create_app, db
from app.models import User, Role
from flask_migrate import Migrate

app = create_app(os.getenv('FLASK_CONFIG') or 'default')
migrate = Migrate(app, db)

@app.shell_context_processor
def make_shell_context():
    return dict(db=db, User=User, Role=Role)
```

**Bài 15: Kiểm thử đơn vị**

Tạo một file `tests/test_basics.py`: *unit tests* với đoạn lệnh sau

```
import unittest
from flask import current_app
from app import create_app, db

class BasicsTestCase(unittest.TestCase):
    def setUp(self):
        self.app = create_app('testing')
        self.app_context = self.app.app_context()
        self.app_context.push()
        db.create_all()
    def tearDown(self):
        db.session.remove()
        db.drop_all()
        self.app_context.pop()

    def test_app_exists(self):
        self.assertFalse(current_app is None)

    def test_app_is_testing(self):
        self.assertTrue(current_app.config['TESTING'])
```

Chạy lệnh khởi chạy thử nghiệm đơn vị

```
@app.cli.command()
def test():
    """Run the unit tests."""
    import unittest
    tests = unittest.TestLoader().discover('tests')
    unittest.TextTestRunner(verbosity=2).run(tests)
```

Các bài kiểm tra đơn vị có thể được thực hiện như sau:

```
(venv) $ flask test
test_app_exists (test_basics.BasicsTestCase) ... ok
test_app_is_testing (test_basics.BasicsTestCase) ... ok
```

```

.-----
Ran 2 tests in 0.001s

OK
-----
```