Manipulação de dados com Flask SQLAlchemy

Experiência Criativa: Criando Soluções Computacionais

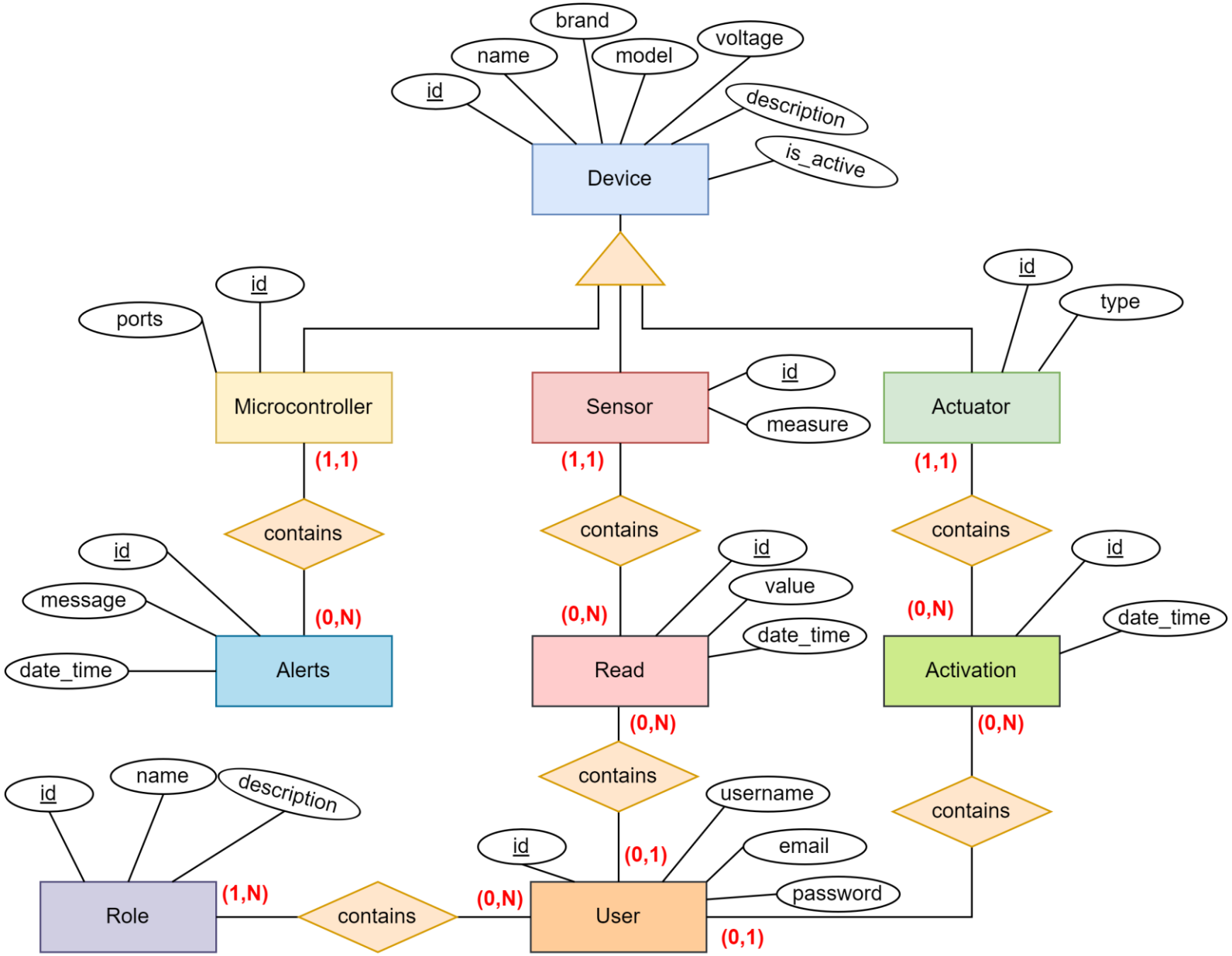
Antonio David Viniski

antonio.david@pucpr.br

Agenda

- Flask SQLAlchemy.
  - Manipulação do Banco de Dados.
    - Inserção de dados.
    - Realização de consultas.
    - Atualização de registros.
    - Exclusão de Registros.
- Flask-Login

Modelo Conceitual - IOT



Inserção Simples

- Para realizar a inserção de dados nas tabelas, precisa-se analisar com atenção os relacionamentos.
  - Inserção na tabela Device.

```
device = Device(name="name", brand="brand", model = "model", voltage = 5,
                    description = "description", is_active = True)
db.session.add(device)
db.session.commit()
```

DML com SQLAlchemy - Insert

- Inserção na tabela **Sensor**, que é um tipo de **Device**.
  - Para inserir na tabela **Sensor**, precisamos primeiro criar um objeto do tipo **Device**, uma vez que um sensor é um tipo de dispositivo, e tem como chave primaria o **id**, que também é chave estrangeira relacionada ao **Device.id**

```
device = Device(name="name", brand="brand", model = "model", voltage = 5,
                description = "description", is_active = True)

db.session.add(device)
db.session.commit()

sensor = Sensor(id = device.id, measure = "measure")
db.session.add(sensor)
db.session.commit()
```

Inserção considerando o relacionamento

- A inserção na tabela **Sensor** também pode ser realiza utilizando o seu relacionamento com a tabela **Device**.

```
device = Device(name="name", brand="brand", model = "model", voltage = 5,
                    description = "description", is_active = True)

sensor = Sensor(id = device.id, measure = "measure")
device.sensors.append(sensor)

db.session.add(device)
db.session.commit()
```

- Dessa forma, é necessária a realização do **commit** uma única vez.
- Esses comandos podem ser adicionados em um método específico da classes Sensor.

Classe Sensor

```
from models.db import db
from models import Device

class Sensor(db.Model):
    __tablename__ = 'sensors'
    id = db.Column('id', db.Integer, db.ForeignKey(Device.id), primary_key=True)
    measure = db.Column(db.String(20))

    def save_sensor(name, brand, model, voltage, description, is_active, measure):
        device = Device(name=name, brand=brand, model = model,
                        voltage = voltage, description = description,
                        is_active = is_active)
        sensor = Sensor(id = device.id, measure = measure)

        device.sensors.append(sensor)

        db.session.add(device)
        db.session.commit()
```

Recuperando dados – Seleção usando os modelos

- Recuperando dados da tabela Device.

```
devices = Device.query.all()
```

- E para recuperar os dados de sensores, os quais são tipos de dispositivos?
  - Devemos utilizar o método **join**, especificando o campo que relaciona as tabelas
  - Além disso, precisamos especificar quais colunas queremos retornar da seleção.
    - para isso utilizamos o método **add_columns;**
    - caso não sejam especificadas, somente dos dados dos sensores estarão disponíveis;

```
sensors = Sensor.query.join(Device, Device.id == Sensor.id)\
                .add_columns(Sensor.id, Device.name, Device.brand, Device.voltage,
                             Device.model, Sensor.measure).all()
```

Da mesma forma, estes comandos podem ser inseridos em um métodos específico da classe Sensor

Classe Sensor

```
from models.db import db
from models import Device

class Sensor(db.Model):
    __tablename__ = 'sensors'
    id = db.Column('id', db.Integer, db.ForeignKey(Device.id), primary_key=True)
    measure = db.Column(db.String(20))

    def save_sensor(name, brand, model, voltage, description, is_active, measure):
        device = Device(name=name, brand=brand, model = model,
                        voltage = voltage, description = description,
                        is_active = is_active)
        sensor = Sensor(id = device.id, measure = measure)

        device.sensors.append(sensor)

        db.session.add(device)
        db.session.commit()

    def get_sensors():
        sensors = Sensor.query.join(Device, Device.id == Sensor.id)\
                .add_columns(Sensor.id, Device.name, Device.brand, Device.voltage,
                             Device.model, Sensor.measure).all()
        return sensors
```

Exercício 1

- Ajustar a classe **Actuator**, adicionando os métodos:
  - save_actuator: insere um novo atuador no banco de dados
  - get_actuators: recupera todos os atuadores do banco de dados.
- Criar o métodos para recuperar:
  - Usuários
  - Leituras
  - Ativações
  - Papéis
  - Etc

Recuperando dados

Seleção usando a sessão do banco de dados

- Recuperando dados dos sensores utilizando a sessão.

```
sensors = db.session.query(Sensor.id, Device.name, Device.brand,
                           Device.voltage, Device.model, Sensor.measure)\
                           .join(Sensor, Sensor.id == Device.id).all()
```

- **sensors** é uma lista de objetos (**dict**), onde cada objeto contém as informações que estão sendo passadas no método query, neste caso: `id, name, brand, voltage, model, measure`.

Podemos passar a lista de sensores diretamente para uma página HTML

Página view_sensors.html

```
{% extends "iot/iot_index.html" %}
{% block content %}
<table id="view_sensors" class="table table-striped table-bordered" style="width:100%">
    <thead>
        <tr>
            <th>Nome</th>
            <th>Modelo</th>
            <th>Marca</th>
            <th>Unidade de Medida</th>
            <th>Tensão de Trabalho</th>
        </tr>
    </thead>
    <tbody>
    {% for device in sensors %}
        <tr>
            <td>{{ device.name }}</td>
            <td>{{ device.brand }}</td>
            <td>{{ device.model }}</td>
            <td>{{ device.measure }}</td>
            <td>{{ device.voltage }}</td>
        </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

Recuperando dados

Seleção de todos os dados dos tabelas

- Recuperando todos os atributos de Device e Sensor utilizando a sessão do banco de dados.

```
sensors = db.session.query(Device, Sensor)\
                    .join(Sensor, Sensor.id == Device.id).all()
```

  - **sensors** é uma lista de tuplas.
  - Cada tupla possui dois objetos: (Device, Sensor).
  - Cada objeto contém as informações de seus respectivos atributos:
    - Device: `id, name, brand, voltage, model, is_active`.
    - Sensor: `id, measure.`

Podemos passar a lista de tuplas diretamente para uma página HTML, no entanto o acesso a essa lista é realizado de forma diferente →

Página view_sensors.html II

```
{% extends "iot/iot_index.html" %}
{% block content %}
<table id="view_sensors" class="table table-striped table-bordered" style="width:100%">
    <thead>
        <tr>
            <th>Nome</th>
            <th>Modelo</th>
            <th>Marca</th>
            <th>Unidade de Medida</th>
            <th>Tensão de Trabalho</th>
        </tr>
    </thead>
    <tbody>
    {% for device, sensor in sensors %}
        <tr>
            <td>{{ device.name }}</td>
            <td>{{ device.brand }}</td>
            <td>{{ device.model }}</td>
            <td>{{ sensor.measure }}</td>
            <td>{{ device.voltage }}</td>
        </tr>
    {% endfor %}
    </tbody>
</table>
{% endblock %}
```

Atualizando registros I

- Para atualizar um registro, é preciso primeiramente verificar se o registro realmente existe.
  - Para fazer verificações, o SQLAlchemy possui o método **filter**:

```
device = Device.query.filter(Device.id == data['id']).first()
```

- Caso a variável **device** não seja nula, podemos efetuar a atualização modificando diretamente os atributos da classe device.

```
device.name = "new name"
device.brand = "new brand"
device.model = "new model"
device.voltage = "new voltage"
device.is_active = "new status"
```

- Para que as alterações sejam salvas no banco de dados precisamos efetuar o commit.

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

Atualizando registros I

- Para atualizar um registro, é preciso primeiramente verificar se o registro realmente existe.
  - Para fazer verificações, o SQLAlchemy possui o método **filter**:

```
device = Device.query.filter(Device.id == data['id']).first()
```

- Caso a variável **device** não seja nula, podemos efetuar a atualização modificando diretamente os atributos da classe device.

```
device.name = "new name"
device.brand = "new brand"
device.model = "new model"
device.voltage = "new voltage"
device.is_active = "new status"
```

- Para que as alterações sejam salvas no banco de dados precisamos efetuar o commit.

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

Excluindo registros

- Para excluir registros o SQLAlchemy possui o método **delete**:
  - Podemos excluir um único registro que satisfaça uma condição:

```
def delete_sensor(id):
    sensor = Sensor.query.filter(Sensor.id == id).first()
    sensor.delete()
    db.session.commit()
```

- Da mesma forma, precisamos excluir o registro de Device associado a esse sensor.
- Podemos também excluir múltiplos registros que satisfaçam uma condição específica

```
def delete_sensor_by_measure(measure):
        Sensor.query.filter_by(measure=measure).delete()
        db.session.commit()
```

Conectando o Flask-SQLAlchemy ao MySQL

Criando a estrutura

- Abrir o MySQL Workbench.

```
CREATE DATABASE restaurant;
CREATE USER test IDENTIFIED BY “test";
GRANT ALL ON *.*  TO test WITH GRANT OPTION;
```

Conexão Flask-SQLAlchemy com o MySQL

- Instalar o módulo de conexão no python com mysql.

```
pip install pymysql
```

- Alterar a string de conexão com o banco no arquivo db

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
#instance = 'sqlite:///restaurant'
instance = "mysql+pymysql://restaurant:r3st2ur4nt@localhost:3306/restaurant"
```

Flask Login

Instalando as dependências

```
pip install flask flask-sqlalchemy flask-login
```

```
python -m pip install flask flask-sqlalchemy flask-login
```

Tutorial Flask Login

- Flask Login:
  - https://www.digitalocean.com/community/tutorials/how-to-add-authentication-to-your-app-with-flask-login
- Roles Required:
  - https://flask-user.readthedocs.io/en/v0.6/roles_required_app.html

Referências

- https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/
- https://jinja.palletsprojects.com/en/3.1.x/
- https://flask.palletsprojects.com/en/2.2.x/