Documentação do Sistema de Biblioteca com SQLAlchemy e Tkinter

Pedro Henrique Valente Favero

September 10, 2024

Contents

1 Introdução

Este documento apresenta o desenvolvimento de um sistema simples de gerenciamento de biblioteca. O sistema foi implementado utilizando as bibliotecas `Tkinter` e `SQLAlchemy` em Python. O Tkinter foi utilizado para a criação

da interface gráfica do usuário (GUI), enquanto o SQLAlchemy foi utilizado para o mapeamento objeto-relacional (ORM) e interação com o banco de dados SQLite.

2 Objetivos

O objetivo deste projeto é criar um sistema que permita:

- Gerenciar livros, autores, gêneros e leitores.
- Registrar e controlar empréstimos de livros.
- Facilitar a criação, leitura, atualização e exclusão (CRUD) de registros no banco de dados de maneira intuitiva.

3 Requisitos do Sistema

Para executar este projeto, você precisará das seguintes ferramentas e bibliotecas:

- Python 3.x
- Tkinter para a interface gráfica.
- SQLAlchemy para a interação com o banco de dados.
- SQLite como banco de dados relacional.

4 Estrutura do Projeto

O projeto foi estruturado em várias partes, como a definição de entidades, funções de CRUD, e a criação da interface gráfica. A seguir, cada parte será explicada em detalhes.

5 Configuração Inicial do Banco de Dados

O SQLAlchemy é utilizado para realizar o mapeamento objeto-relacional, permitindo a criação e gerenciamento do banco de dados sem a necessidade de escrever SQL manualmente. A primeira parte do código envolve a configuração do SQLAlchemy e a definição das tabelas necessárias.

5.1 Definição das Entidades

As classes `Livro`, `Autor`, `Genero`, e `Leitor` representam as entidades no banco de dados. Essas classes mapeiam diretamente para tabelas no banco de dados SQLite. Abaixo está a definição das entidades e seus relacionamentos.

```
from sqlalchemy import create_engine, Column, Integer, String
    , ForeignKey, Date, Table
from sqlalchemy.orm import declarative_base, relationship,
    sessionmaker
from datetime import date

Base = declarative_base()

# Tabela de associação para o relacionamento muitos-para-
    muitos
livro_leitor = Table('livro_leitor', Base.metadata,
    Column('livro_id', Integer, ForeignKey('livros.id')),
    Column('leitor_id', Integer, ForeignKey('leitores.id')),
    Column('data_emprestimo', Date, default=date.today())
)

# Classe Livro
class Livro(Base):
    __tablename__ = 'livros'
    id = Column(Integer, primary_key=True)
    titulo = Column(String, nullable=False)
    isbn = Column(String, unique=True, nullable=False)
    autor_id = Column(Integer, ForeignKey('autores.id'))
    genero_id = Column(Integer, ForeignKey('generos.id'))

    autor = relationship("Autor", back_populates="livros")
    genero = relationship("Genero", back_populates="livros")
    leitores = relationship("Leitor", secondary=livro_leitor,
        back_populates="livros")
```

5.2 Relacionamentos entre Entidades

Os relacionamentos entre as entidades são definidos usando o SQLAlchemy, permitindo que as tabelas sejam vinculadas. A tabela `livro_leitor` é uma tabela de associação que gerencia o relacionamento muitos-para-muitos entre os livros e os leitores. Um livro pode ser emprestado por vários leitores e um leitor pode emprestar vários livros.

- `Livro`: Possui relacionamentos com as entidades `Autor`, `Genero` e `Leitor`.
- `Autor`: Pode ter vários livros associados.

- `Genero`: Classifica os livros e tem um relacionamento um-para-muitos com `Livro`.
- `Leitor`: Está relacionado a vários livros através de empréstimos.

6 Operações CRUD

O sistema inclui funções para adicionar, listar, atualizar e remover registros para as entidades `Livro`, `Autor`, `Genero`, e `Leitor`. As funções CRUD são essenciais para garantir a manipulação dos dados de forma eficiente.

6.1 Exemplo de Funções CRUD para Livros

```
# Adicionar um novo livro ao banco de dados
def adicionar_livro(titulo, isbn, autor_id, genero_id):
    novo_livro = Livro(titulo=titulo, isbn=isbn, autor_id=
        autor_id, genero_id=genero_id)
    session.add(novo_livro)
    session.commit()

# Listar todos os livros registrados
def listar_livros():
    return session.query(Livro).all()
```

Essas funções são utilizadas pela interface gráfica para interagir com o banco de dados, permitindo que os dados sejam exibidos e modificados de maneira fácil.

7 Interface Gráfica com Tkinter

O `Tkinter` é utilizado para criar a interface gráfica do usuário (GUI). A interface permite que o usuário adicione, visualize, atualize e remova registros das entidades, além de gerenciar os empréstimos de livros.

7.1 Gerenciamento de Livros

A função `abrir_janela_livros` cria a janela de gerenciamento de livros. Nessa janela, o usuário pode adicionar novos livros, remover livros existentes e atualizar informações dos livros. Além disso, uma tabela mostra todos os livros cadastrados.

```
def abrir_janela_livros():
    livro_window = tk.Toplevel()
    livro_window.title("Gerenciar Livros")

    def atualizar_lista_livros():
        livros = listar_livros()
        lista_livros.delete(*lista_livros.get_children())
        for livro in livros:
            autor = livro.autor.nome if livro.autor else "
                Desconhecido"
            genero = livro.genero.nome if livro.genero else "
                Desconhecido"
            lista_livros.insert("", "end", values=(livro.id,
                livro.titulo, livro.isbn, autor, genero))

    # Fun   o para salvar um novo livro
    def salvar_livro():
        titulo = entry_titulo.get()
        isbn = entry_isbn.get()
        autor_id = combo_autor.get().split('-')[0].strip()
        genero_id = combo_genero.get().split('-')[0].strip()
        if titulo and isbn and autor_id and genero_id:
            adicionar_livro(titulo, isbn, autor_id, genero_id
                )
            messagebox.showinfo("Sucesso", "Livro adicionado
                com sucesso!")
            atualizar_lista_livros()
        else:
            messagebox.showerror("Erro", "Preencha todos os
                campos.")
```

7.2 Janela de Empréstimos

Outra parte importante do sistema é o gerenciamento de empréstimos. O usuário pode registrar um empréstimo, remover ou atualizar empréstimos existentes. A janela exibe todos os empréstimos registrados, com o nome do leitor e os livros que ele pegou emprestado.

```
def abrir_janela_emprestimos():
    emprestimo_window = tk.Toplevel()
    emprestimo_window.title("Gerenciar Empr stimos")

    def atualizar_lista_emprestimos():
        leitores = listar_emprestimos()
        lista_emprestimos.delete(*lista_emprestimos.
            get_children())
        for leitor in leitores:
```

```
            livros = ", ".join([livro.titulo for livro in
                leitor.livros])
            lista_emprestimos.insert("", "end", values=(
                leitor.id, leitor.nome, livros))

    # Registrar um novo empr stimo
    def registrar():
        leitor_id = combo_leitor.get().split('-')[0].strip()
        livro_id = combo_livro.get().split('-')[0].strip()
        if leitor_id and livro_id:
            registrar_emprestimo(livro_id, leitor_id)
            messagebox.showinfo("Sucesso", "Empr stimo
                registrado com sucesso!")
            atualizar_lista_emprestimos()
        else:
            messagebox.showerror("Erro", "Selecione um leitor
                e um livro.")
```

8 Considerações Finais

Este sistema de gerenciamento de biblioteca foi projetado para ser simples, porém eficiente, utilizando as ferramentas `Tkinter` para a interface gráfica e `SQLAlchemy` para gerenciar o banco de dados. Todas as funcionalidades básicas de um sistema de gerenciamento de biblioteca estão presentes, incluindo o gerenciamento de livros, autores, leitores e empréstimos.

O código foi estruturado de forma modular, facilitando a manutenção e adição de novas funcionalidades no futuro.