Aléxis Santos, Hetal Verma

Curso Técnico de Gestão e Programação de Sistemas Informáticos

Escola Profissional Bento de Jesus Caraça, Delegação do Barreiro

P.S.I.: Programação e Sistemas de Informação

Professor André Rolo

3 de fevereiro de 2025

Índice de Ilustrações

Introdução

Neste projeto, foi desenvolvido um programa em *Python* no qual foram implementadas operações *CRUD* (*Create, Read, Update, Delete*) sobre uma base de dados relacional. O tema escolhido baseou-se nos campeões do jogo *League of Legends*. A base de dados foi estruturada para armazenar informações relevantes sobre os campeões, incluindo o nome, tipo de dano e posição. A estrutura do projeto foi organizada em duas pastas principais: *sqlite_db*, onde a base de dados foi armazenada, e *src*, que contém o código-fonte dividido de acordo com as operações *CRUD*. Cada operação foi implementada separadamente, garantindo uma estrutura modular e eficiente.

Objetivos do Projeto

O principal objetivo deste projeto foi a implementação de um sistema de gestão de dados utilizando *Python* e *SQLite3*, permitindo a execução de operações *CRUD* sobre informações relacionadas aos campeões do *League of Legends*. A implementação seguiu boas práticas de programação, garantindo modularidade, estruturação clara do código e facilidade de manutenção.

A Aplicação

Este projeto consiste na criação de uma aplicação desenvolvida em *Python*, que utiliza uma base de dados *SQLite* para armazenar e gerir informação sobre os campeões do jogo *League of Legends*. Para isso, são implementadas operações CRUD (*Create, Read, Update* e *Delete*), que permitem manipular os dados armazenados.

A base de dados está estruturada de forma a conter três tabelas dedicadas aos campeões, onde estão organizadas as seguintes colunas:

- *champs* = id, nome
- *champs_dmg* = id, nome, *dmg*
- *champs_lane* = id, nome, *lane*

Estrutura do projeto

A aplicação está organizada em duas diretorias principais:

- *sqlite_db* - Diretório que contém a base de dados *epbjc.db*.
- *src* - Diretório que contém os scripts necessários para efetuar operações sobre a base de dados. Os ficheiros estão organizados em subdiretórios, consoante a sua funcionalidade:

create/ - Contém os scripts responsáveis pela criação de tabelas e inserção de novos dados.

read/ - Inclui os scripts que permitem consultar e extrair informação da base de dados.

update/ - Inclui os scripts que permitem a atualização de dados previamente registados.

***Queries* Usados**

- **Criação da Base de Dados e Tabelas:**

Inicialmente, a base de dados foi criada e as tabelas foram definidas utilizando o comando *CREATE TABLE*:

Figura 1

```
#importar a biblioteca sqlite3
import sqlite3

#estabelecer a conexao com a base de dados
conex = sqlite3.connect('C:\\Users\\solan\\Desktop\\projetom15\\projeto_modulo_15\\sqlite_db\\sqlite.db')
cursor = conex.cursor()

#criar tabela champs
cursor.execute('''
CREATE TABLE IF NOT EXISTS champs (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    nome TEXT NOT NULL
)
''')
```

Neste código, são criadas três tabelas distintas que armazenam os campeões, seus tipos de dano e suas posições no jogo. Caso a base de dados já exista, a função *IF NOT EXISTS* impede a recriação das tabelas, evitando perda de dados.

- **Inserção de dados:**

Para preencher a base de dados com alguns campeões, foram usados comandos *INSERT INTO*:

Figura 2

```
#inserir registos na tabela champs
cursor.execute('INSERT INTO champs (nome) VALUES ("Akali")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Katarina")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Cassiopeia")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Sion")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Jarvan")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Rengar")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Ahri")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Sett")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Rakan")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Xayah")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Soraka")')
cursor.execute('INSERT INTO champs (nome) VALUES ("Yuumi")')


#confirmar alteracoes
conex.commit()
#fechar conexao
conex.close()
```

Essa função recebe os atributos de um campeão e insere os valores correspondentes nas tabelas apropriadas.

- **Leitura de dados:**

  Para visualizar os campeões armazenados, utilizou-se *SELECT*:

Figura 3

```
pergunta = input("Quer ler a união entre a tabela do dano e dos champions(1), ou apenas a do champions(2)?")
if pergunta == "1":
    #extrair todas as colunas da tabela champs e da tabela champs_dmg, e une as colunas onde os nomes coincidem
    cursor.execute('SELECT DISTINCT * FROM champs INNER JOIN champs_dmg ON champs.nome = champs_dmg.champ_nome')
    resultados = cursor.fetchall()
elif pergunta == "2":
    #extrair todas as colunas da tabela champs
    cursor.execute('SELECT * FROM champs')
    resultados = cursor.fetchall()
else:
    #caso a opcao inserida pelo utilizador nao exista
    print("Opção inválida")

#imprime todas as colunas, uma a uma, com o ciclo for
for champ in resultados:
    print(champ)

#confirmar alteracoes
conex.commit()
#encerrar conexoes
conex.close()
```

- **Atualização de dados:**

Caso fosse necessário atualizar alguma informação, como mudar a lane de um campeão, utilizou-se *UPDATE*:

Figura 4

```
#atualiza a coluna nome onde o id é um
cursor.execute('''UPDATE champs SET nome = "Taric" WHERE id = 1''')

#confirmar alteracoes
conex.commit()
#encerrar conexao
conex.close()
```

- **Remoção de dados**:

Para remover registos da base de dados, o comando *DELETE FROM* foi utilizado:

Figura 5

```
#importar biblioteca sqlite3
import sqlite3

#estabelecer conexao
conex = sqlite3.connect('C:\\Users\\solan\\Desktop\\projetom15\\projeto_modulo_15\\sqlite_db\\sqlite.db')
cursor = conex.cursor()

#confirmacoes de mau gosto
pergunta = input("Quer apagar a tabela champs? ")
if pergunta == "sim":
    crtz = input("Tem a certeza? ")
    if crtz == "sim":
        absoluta = input("Tem a certeza absoluta? ")
        if absoluta == "sim":
            warning = input("Cuidado que vai tudo á vida, quer MESMO? ")
            if warning == "sim":
                #apagar tabela champs
                cursor.execute('DELETE FROM champs')

if pergunta or crtz or absoluta or warning == "nao":
    print("Obrigada meu")

#confirmar alteracoes
conex.commit()
#encerrar conexao
conex.close()
```

- **Uso do *INNER JOIN*:**

  Com o objetivo de aumentar a organização da base de dados, uma segunda tabela foi criada para armazenar informações adicionais:

Figura 6

```
#importar biblioteca sqlite3
import sqlite3

#estabelecer conexao a db
conex = sqlite3.connect('C:\\Users\\solan\\Desktop\\projetom15\\projeto_modulo_15\\sqlite_db\\sqlite.db')
cursor_dmg = conex.cursor()

#criar tabela champs_dmg
cursor_dmg.execute('''
CREATE TABLE IF NOT EXISTS champs_dmg (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    champ_nome TEXT NOT NULL,
    dmg TEXT NOT NULL,
    FOREIGN KEY (champ_nome) REFERENCES champs (nome)
)
''')
```

A relação entre as tabelas foi estabelecida pela chave estrangeira *Champ_nome*, que referencia o nome da tabela "campeões". Para combinar dados das duas tabelas, o comando *INNER JOIN* foi aplicado:

Figura 7

```
pergunta = input("Quer ler a união entre a tabela do dano e dos champions(1), ou apenas a do champions(2)?")
if pergunta == "1":
    #extrair todas as colunas da tabela champs e da tabela champs_dmg, e une as colunas onde os nomes coincidem
    cursor.execute('SELECT DISTINCT * FROM champs INNER JOIN champs_dmg ON champs.nome = champs_dmg.champ_nome')
    resultados = cursor.fetchall()
elif pergunta == "2":
    #extrair todas as colunas da tabela champs
    cursor.execute('SELECT * FROM champs')
    resultados = cursor.fetchall()
else:
    #caso a opcao inserida pelo utilizador nao exista
    print("Opção inválida")

#imprime todas as colunas, uma a uma, com o ciclo for
for champ in resultados:
    print(champ)

#confirmar alteracoes
conex.commit()
#encerrar conexoes
conex.close()
```

Funcionalidades do sistema

- Inserção de novos campeões.
- Consulta de campeões por nome, tipo de dano ou posição.
- Atualização de informações de campeões existentes.
- Remoção de campeões do banco de dados.

Conclusão

Este projeto foi muito útil para aprender mais sobre bases de dados relacionais e como usá-las com *Python*. Algumas partes foram mais fáceis, como criar as tabelas e inserir os dados, porque o SQLite3 tem uma sintaxe simples. Além disso, separar as operações CRUD em ficheiros diferentes ajudou a manter o código mais organizado e fácil de entender.

Por outro lado, algumas dificuldades apareceram, como garantir que os dados estivessem sempre corretos entre as tabelas e evitar erros ao atualizar ou remover registos. Foi preciso ter atenção extra para evitar problemas e garantir que tudo funcionasse como esperado. Também foi um desafio estruturar bem o código para que ficasse claro e reutilizável, mas no final, isso tornou o projeto mais eficiente.

No geral, este projeto ajudou a perceber como bases de dados funcionam na prática e a importância de planear bem antes de programar. Apesar das dificuldades, o resultado foi positivo e o conhecimento adquirido será muito útil para futuros projetos mais complexos.