**Documentação Sistema - Livraria Online Lima**

Disciploina: Laboratório de Desenvolvimento de Software - SAGAH

ESAB - Escola Superior Aberta do Brasil

**Indice**

1 - Código: main.py

2 - Código: database.py

1. **CÓDIGO: MAIN.PY**

from typing import Optional

import PySide6.QtCore

from PySide6 import QtCore

from PySide6.QtGui import QIcon

from PySide6.QtWidgets import (QApplication, QFileDialog, QMainWindow, QWidget, QMessageBox, QTableWidgetItem, QTreeWidgetItem)

from ui\_login import Ui\_Login

from ui\_main import Ui\_MainWindow

import sys

from database import DataBase

import sqlite3

import pandas as pd

from PySide6.QtSql import QSqlDatabase, QSqlTableModel

import re

from datetime import date

from datetime import timedelta

numero\_id\_usuario = 0

global\_id\_usuario = 0

nome\_id\_usuario = 0

###\_Login\_#########################################################################################################################

class Login(QWidget, Ui\_Login):

    def \_\_init\_\_(*self*) -> None:

        super(Login, *self*).\_\_init\_\_()

*self*.tentativas = 0

*self*.setupUi(*self*)

*self*.setWindowTitle("Livraria OnLine Lima")

        appIcon = QIcon('C:/Users/SAMSUNG/Documents/QtPy Biblioteca/Imagens/icone.jpg')

*self*.setWindowIcon(appIcon)

*self*.btn\_login.clicked.connect(*self*.checkLogin)

*self*.btn\_cadastrase.clicked.connect(*self*.open\_new\_user)

    def open\_new\_user(*self*):

*self*.users = DataBase()

*self*.users.conecta()

        autenticado = "Novo"

        nomeuser = "Novo"

*self*.w = MainWindow(nomeuser, autenticado)

*self*.w.show()

*self*.close()

    def checkLogin(*self*):

*self*.users = DataBase()

*self*.users.conecta()

        autenticado = *self*.users.check\_user(*self*.txt\_user.text().upper(), *self*.txt\_password.text())

        if autenticado == "administrador"  or autenticado == "user":

*self*.w = MainWindow(*self*.txt\_user.text(), autenticado)

*self*.w.show()

*self*.close()

        else:

            if *self*.tentativas < 3:

                msg = QMessageBox()

                msg.setIcon(QMessageBox.Warning)

                msg.setWindowTitle("Erro ao Acessar")

                msg.setText(f'Login ou senha incorreto \n \n Tentativa: {*self*.tentativas +1} de 3')

                msg.exec\_()

*self*.tentativas += 1

            if *self*.tentativas == 3:

*self*.users.close\_connection()

                sys.exit(0)

###################################################################################################################################

###\_Sistema\_#######################################################################################################################

class MainWindow(QMainWindow, Ui\_MainWindow):

    def \_\_init\_\_(*self*, *nomeuser*, *user*):

        super(MainWindow, *self*).\_\_init\_\_()

*self*.setupUi(*self*)

*self*.setWindowTitle("Livraria OnLine Lima")

*self*.user = *user*

        appIcon = QIcon('C:/Users/SAMSUNG/Documents/QtPy Biblioteca/Imagens/icone.jpg')

*self*.setWindowIcon(appIcon)

        if *user* ==  "administrador":

*self*.btn\_novo.setVisible(False)

        if *user* ==  "user":

*self*.btn\_novo.setVisible(False)

*self*.btn\_pg\_cadastro.setVisible(False)

*self*.btn\_pg\_cadastro\_livro.setVisible(False)

*self*.btn\_pg\_relatorio\_locacao.setVisible(False)

        if *user* == "Novo":

*self*.btn\_pg\_cadastro.setVisible(False)

*self*.btn\_pg\_cadastro\_livro.setVisible(False)

*self*.btn\_home.setVisible(False)

*self*.btn\_user.setVisible(False)

*self*.btn\_locacao.setVisible(False)

*self*.btn\_catalogo.setVisible(False)

*self*.btn\_devolucao.setVisible(False)

*self*.pg\_home.setVisible(False)

*self*.btn\_pg\_relatorio\_locacao.setVisible(False)

*self*.pg\_cadastro\_user.setVisible(True)

*self*.cb\_perfil.setVisible(False)

*self*.label\_13.setVisible(False)

###\_Navegação Páginas\_#############################################################################################################

*self*.btn\_home.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pg\_home))

*self*.btn\_user.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pg\_user))

*self*.btn\_catalogo.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pg\_catalogo))

*self*.btn\_locacao.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pg\_locacao))

*self*.btn\_devolucao.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pg\_devolucao))

*self*.btn\_pg\_cadastro.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pg\_cadastro\_user))

*self*.btn\_pg\_cadastro\_livro.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pag\_cadastro\_livro))

*self*.btn\_pg\_relatorio\_locacao.clicked.connect(lambda: *self*.Pages.setCurrentWidget(*self*.pg\_relatorio))

###\_Botões\_#######################################################################################################################

*self*.btn\_cadastrar.clicked.connect(*self*.subscribe\_user)

*self*.btn\_cadastrar\_livro.clicked.connect(*self*.subscribe\_livros\_locacao)

*self*.btn\_cadastrar\_livro.clicked.connect(*self*.subscribe\_livros)

*self*.btn\_atualizar\_livro.clicked.connect(*self*.atualiza\_livro\_adm)

*self*.btn\_excluir\_livro.clicked.connect(*self*.deletar\_livro)

*self*.btn\_cadastrar\_2.clicked.connect(*self*.salvar\_edicao\_user)

*self*.txt\_nome\_3.editingFinished.connect(*self*.edita\_user)

*self*.btn\_gerar\_locacao.clicked.connect(*self*.gerar\_saida)

*self*.btn\_efetivar\_devolucao.clicked.connect(*self*.gerar\_devolucao)

*self*.btn\_estornar.clicked.connect(*self*.gerar\_estorno)

*self*.btn\_finalizar\_locacao.clicked.connect(*self*.finalizar\_locacao)

###\_Listagem\_#######################################################################################################################

*self*.label\_31.setText(*nomeuser*)

*self*.limpar\_status()

*self*.reset\_table()

        if *user* != "Novo":

*self*.global\_user()

*self*.txt\_filter\_catalogo.textChanged.connect(*self*.filtro\_catalogo)

###################################################################################################################################

###\_Usuário\_#######################################################################################################################

    def global\_user (*self*):

        global global\_id\_usuario

        global nome\_id\_usuario

        db = DataBase()

        db.conecta()

        campos = db.busca\_user(*self*.label\_31.text())

        global\_id\_usuario = campos[5]

        nome\_id\_usuario = campos[1]

        db.close\_connection()

    def subscribe\_user(*self*):

        if *self*.txt\_senha.text() != *self*.txt\_senha2.text():

            msg = QMessageBox()

            msg.setIcon(QMessageBox.Warning)

            msg.setWindowTitle("Senhas Divergentes")

            msg.setText("A senha não é igual")

            msg.exec\_()

            return None

        name = *self*.txt\_nome.text()

        user = *self*.txt\_usuario.text()

        password = *self*.txt\_senha.text()

        access = *self*.cb\_perfil.currentText()

        email = *self*.txt\_email.text()

        tel = *self*.txt\_tel.text()

        db = DataBase()

        db.conecta()

        db.inser\_user(name, user, password, access, email, tel)

        db.close\_connection()

        msg = QMessageBox()

        msg.setIcon(QMessageBox.Warning)

        msg.setWindowTitle("Cadastro de usuário")

        msg.setText("Cadastro realizado com sucesso")

        msg.exec\_()

*self*.txt\_nome.setText("")

*self*.txt\_usuario.setText("")

*self*.txt\_senha.setText("")

*self*.txt\_senha2.setText("")

*self*.txt\_email.setText("")

*self*.txt\_tel.setText("")

    def edita\_user (*self*):

        global numero\_id\_usuario

        global global\_id\_usuario

        db = DataBase()

        db.conecta()

        campos = db.busca\_user(*self*.txt\_nome\_3.text())

        if campos == None:

            db.close\_connection()

            msg = QMessageBox()

            msg.setIcon(QMessageBox.Warning)

            msg.setWindowTitle("Erro Usuário")

            msg.setText("Usuário inexistente")

            msg.exec\_()

*self*.txt\_nome\_2.setText("")

*self*.txt\_usuario\_2.setText("")

*self*.txt\_email\_2.setText("")

*self*.txt\_tel\_2.setText("")

*self*.txt\_senha\_2.setText("")

*self*.txt\_senha2\_2.setText("")

        else:

            numero\_id\_usuario = campos[5]

        if numero\_id\_usuario == global\_id\_usuario:

*self*.txt\_nome\_2.setText(campos[0])

*self*.txt\_usuario\_2.setText(campos[1])

*self*.txt\_senha\_2.setText(campos[2])

*self*.txt\_senha2\_2.setText(campos[2])

*self*.txt\_email\_2.setText(campos[3])

*self*.txt\_tel\_2.setText(campos[4])

        else:

            db.close\_connection()

            msg = QMessageBox()

            msg.setIcon(QMessageBox.Warning)

            msg.setWindowTitle("Erro Usuário")

            msg.setText("Informe o nome do usuário logado")

            msg.exec\_()

*self*.txt\_nome\_2.setText("")

*self*.txt\_usuario\_2.setText("")

*self*.txt\_email\_2.setText("")

*self*.txt\_tel\_2.setText("")

*self*.txt\_senha\_2.setText("")

*self*.txt\_senha2\_2.setText("")

    def salvar\_edicao\_user (*self*):

        global numero\_id\_usuario

        dados= []

        nome = *self*.txt\_nome\_2.text()

        usuario = *self*.txt\_usuario\_2.text()

        email = *self*.txt\_email\_2.text()

        telefone = *self*.txt\_tel\_2.text()

        senha = *self*.txt\_senha\_2.text()

        dados = [[numero\_id\_usuario,nome,usuario,senha,email,telefone]]

        db = DataBase()

        db.conecta()

        for emp in dados:

            db.update\_user(tuple(emp))

        db.close\_connection()

        msg = QMessageBox()

        msg.setIcon(QMessageBox.Warning)

        msg.setWindowTitle("Atualização de usuário")

        msg.setText("Atualização realizada com sucesso")

        msg.exec\_()

*self*.txt\_nome\_2.setText("")

*self*.txt\_usuario\_2.setText("")

*self*.txt\_email\_2.setText("")

*self*.txt\_tel\_2.setText("")

*self*.txt\_senha\_2.setText("")

*self*.txt\_senha2\_2.setText("")

##################################################################################################################################

###\_Livros\_#######################################################################################################################

    def subscribe\_livros(*self*):

        db = DataBase()

        db.conecta()

        status\_livro = ("Disponível")

        fullDataSet = (*self*.txt\_nome\_livro.text(), *self*.txt\_genero\_livro.text(), *self*.txt\_autor\_livro.text(), *self*.txt\_descricao\_livro.text(), status\_livro, *self*.txt\_quantidade\_livro.text())

        resp = db.inser\_livro(fullDataSet)

*self*.txt\_nome\_livro.setText("")

*self*.txt\_genero\_livro.setText("")

*self*.txt\_autor\_livro.setText("")

*self*.txt\_descricao\_livro.setText("")

*self*.txt\_quantidade\_livro.setText("")

*self*.listar\_livros\_adm()

        if resp == "ok":

            msg = QMessageBox()

            msg.setIcon(QMessageBox.Warning)

            msg.setWindowTitle("Cadastro de Livros")

            msg.setText("Cadastro realizado com sucesso")

            msg.exec()

            db.close\_connection()

            return

        else:

            msg = QMessageBox()

            msg.setIcon(QMessageBox.critical)

            msg.setWindowTitle("Erro")

            msg.setText("Erro ao cadastrar")

            msg.exec()

            db.close\_connection()

            return

    def subscribe\_livros\_locacao(*self*):

        db = DataBase()

        db.conecta()

        status\_livro = ("Disponível")

        data\_locacao = ("99/99/9999")

        data\_prevista\_devolucao = ("99/99/9999")

        data\_devolucao = ("99/99/9999")

        usuario = ("sistema")

        fullDataSet = (*self*.txt\_nome\_livro.text(), status\_livro, *self*.txt\_quantidade\_livro.text(), data\_locacao, data\_prevista\_devolucao, data\_devolucao, usuario)

        resp = db.inser\_livro\_locacao(fullDataSet)

        if resp == "ok":

            db.close\_connection()

            return

        else:

            db.close\_connection()

            return

    def listar\_livros\_adm (*self*):

        db = DataBase()

        db.conecta()

        result = db.select\_all\_livros()

*self*.tw\_livros.clearContents()

*self*.tw\_livros.setRowCount(len(result))

        for row, text in enumerate(result):

            for column, data in enumerate(text):

*self*.tw\_livros.setItem(row, column, QTableWidgetItem(str(data)))

        db.close\_connection()

    def atualiza\_livro\_adm (*self*):

        dados = []

        update\_dados = []

        for row in range(*self*.tw\_livros.rowCount()):

            for column in range(*self*.tw\_livros.columnCount()):

                dados.append(*self*.tw\_livros.item(row, column).text())

            update\_dados.append(dados)

            dados = []

        db = DataBase()

        db.conecta()

        for emp in update\_dados:

            db.update\_livro(tuple(emp))

        db.close\_connection()

        msg = QMessageBox()

        msg.setIcon(QMessageBox.Warning)

        msg.setWindowTitle("Atualização de Livros")

        msg.setText("Atualização realizada com sucesso")

        msg.exec\_()

*self*.tw\_livros.reset()

*self*.listar\_livros\_adm()

    def deletar\_livro (*self*):

        db = DataBase()

        db.conecta()

        msg = QMessageBox()

        msg.setWindowTitle("Excluir")

        msg.setText("esse registro será excluído.")

        msg.setInformativeText("Você tem certeza que deseja excluir?")

        msg.setStandardButtons(QMessageBox.Yes | QMessageBox.No)

        resp = msg.exec()

        if resp == QMessageBox.Yes:

            id\_livro = *self*.tw\_livros.selectionModel().currentIndex().siblingAtColumn(0).data()

            result = db.delete\_livros(id\_livro)

*self*.listar\_livros\_adm()

            msg = QMessageBox()

            msg.setIcon(QMessageBox.Warning)

            msg.setWindowTitle("Excluir Livro")

            msg.setText(result)

            msg.exec()

        db.close\_connection()

###################################################################################################################################

####\_tabelas\_QT\_###################################################################################################################

    def livro\_saida(*self*):

        cn = sqlite3.connect('system.db')

        result = pd.read\_sql\_query("SELECT \* FROM livros WHERE status\_livro = 'Disponível'", cn)

        result = result.values.tolist()

*self*.x = ""

        for i in result:

*self*.campo = QTreeWidgetItem(*self*.tw\_livro\_saida, i)

*self*.campo.setCheckState(0, QtCore.Qt.*CheckState*.Unchecked)

*self*.x = i[0]

*self*.tw\_livro\_saida.setSortingEnabled(True)

        for i in range(0,6):

*self*.tw\_livro\_saida.resizeColumnToContents(i)

    def saida(*self*):

        cn = sqlite3.connect('system.db')

        result = pd.read\_sql\_query("SELECT id\_livro\_lo, nome\_livro\_lo, quantidade\_livro\_lo, data\_locacao\_lo, data\_prevista\_devolucao\_lo, usuario\_lo  FROM locacao WHERE status\_livro\_lo = 'Reservado'",  cn)

        result = result.values.tolist()

*self*.x = ""

        for i in result:

*self*.campo = QTreeWidgetItem(*self*.tw\_saida, i)

*self*.campo.setCheckState(0, QtCore.Qt.*CheckState*.Unchecked)

*self*.x = i[0]

*self*.tw\_saida.setSortingEnabled(True)

        for i in range(0,6):

*self*.tw\_saida.resizeColumnToContents(i)

    def tabela\_catalogo(*self*):

        db = QSqlDatabase("QSQLITE")

        db.setDatabaseName("system.db")

        db.open()

*self*.model = QSqlTableModel(*db*=db)

*self*.tv\_livros.setModel(*self*.model)

*self*.model.setTable("livros")

*self*.model.select()

    def filtro\_catalogo(*self*, *s*):

*s* = re.sub("[\W\_]+", "", *s*)

        filter\_str = 'nome\_livro LIKE "%{}%"'.format(*s*)

*self*.model.setFilter(filter\_str)

    def reset\_table(*self*):

*self*.tw\_livro\_saida.clear()

*self*.tw\_saida.clear()

*self*.tw\_devolucao.clear()

*self*.listar\_livros\_adm()

*self*.livro\_saida()

*self*.saida()

*self*.tabela\_catalogo()

*self*.livro\_devolucao()

*self*.relat\_locado()

    def gerar\_saida(*self*):

*self*.checked\_item\_out = []

        def recurse(*parent\_item*):

            for i in range(*parent\_item*.childCount()):

                child = *parent\_item*.child(i)

                grand\_children = child.childCount()

                if grand\_children == 0:

                    recurse(child)

                if child.checkState(0) != QtCore.Qt.*CheckState*.Unchecked:

*self*.checked\_item\_out.append(child.text(1))

        recurse(*self*.tw\_livro\_saida.invisibleRootItem())

*self*.question('saida')

    def gerar\_estorno(*self*):

*self*.checked\_items = []

        def recurse(*parent\_item*):

            for i in range(*parent\_item*.childCount()):

                child = *parent\_item*.child(i)

                grand\_children = child.childCount()

                if grand\_children == 0:

                    recurse(child)

                if child.checkState(0) != QtCore.Qt.*CheckState*.Unchecked:

*self*.checked\_items.append(child.text(1))

        recurse(*self*.tw\_saida.invisibleRootItem())

*self*.question('estorno')

    def question(*self*, *table*):

        msgBox = QMessageBox()

        if *table* == 'estorno':

            msgBox.setText("Deseja Estornar a Transação?")

            msgBox.setInformativeText("Os livros selecionados voltarão para a lista de seleção \n clique em Yes para Confirmar")

            msgBox.setStandardButtons(QMessageBox.Yes | QMessageBox.No)

            msgBox.setDetailedText(f"livros: {*self*.checked\_items}")

        else:

            msgBox.setText("Deseja realizar a locação?")

            msgBox.setInformativeText("Os livros selecionados irão para o registro de locação \n clique em Yes para Confirmar")

            msgBox.setStandardButtons(QMessageBox.Yes | QMessageBox.No)

            msgBox.setDetailedText(f"livros: {*self*.checked\_item\_out}")

        msgBox.setIcon(QMessageBox.Question)

        ret = msgBox.exec()

        if ret == QMessageBox.Yes:

            if *table* == "estorno":

*self*.db = DataBase()

*self*.db.conecta()

*self*.db.estorno\_disp\_livro(*self*.checked\_items)

*self*.db.estorno\_disp\_livro\_locacao(*self*.checked\_items)

*self*.db.close\_connection()

*self*.reset\_table()

            else:

                data\_locacao = date.today()

                td = timedelta(*days*=60)

                data\_prevista\_devolucao = data\_locacao + td

                data\_locacao = data\_locacao.strftime('%d/%m/%y')

                data\_prevista\_devolucao = data\_prevista\_devolucao.strftime('%d/%m/%y')

                usuario = nome\_id\_usuario

*self*.db = DataBase()

*self*.db.conecta()

*self*.db.update\_disp\_livro(*self*.checked\_item\_out)

*self*.db.update\_disp\_livro\_locacao(data\_locacao, data\_prevista\_devolucao, usuario, *self*.checked\_item\_out)

*self*.db.close\_connection()

*self*.reset\_table()

    def limpar\_status(*self*):

        db = DataBase()

        db.conecta()

        db.status\_limpo()

        db.close\_connection()

*self*.reset\_table()

    def finalizar\_locacao(*self*):

        db = DataBase()

        db.conecta()

        db.confirmar\_locacao()

        db.close\_connection()

*self*.reset\_table()

        msg = QMessageBox()

        msg.setIcon(QMessageBox.Warning)

        msg.setWindowTitle("Finalizar Locação")

        msg.setText("Locação realizada com sucesso")

        msg.exec\_()

    def livro\_devolucao(*self*):

*self*.global\_user()

        nome = nome\_id\_usuario

        cn = sqlite3.connect('system.db')

        result = pd.read\_sql\_query(f"""SELECT

                                   id\_livro\_lo, nome\_livro\_lo,

                                   quantidade\_livro\_lo,

                                   data\_locacao\_lo,

                                   data\_prevista\_devolucao\_lo,

                                   usuario\_lo,

                                   status\_livro\_lo

                                   FROM locacao WHERE usuario\_lo = '{nome}' AND status\_livro\_lo = 'Locado' """, cn)

        result = result.values.tolist()

*self*.x = ""

        for i in result:

*self*.campo = QTreeWidgetItem(*self*.tw\_devolucao, i)

*self*.campo.setCheckState(0, QtCore.Qt.*CheckState*.Unchecked)

*self*.x = i[0]

*self*.tw\_devolucao.setSortingEnabled(True)

        for i in range(0,6):

*self*.tw\_devolucao.resizeColumnToContents(i)

    def gerar\_devolucao(*self*):

*self*.checked\_item\_out = []

        def recurse(*parent\_item*):

            for i in range(*parent\_item*.childCount()):

                child = *parent\_item*.child(i)

                grand\_children = child.childCount()

                if grand\_children == 0:

                    recurse(child)

                if child.checkState(0) != QtCore.Qt.*CheckState*.Unchecked:

*self*.checked\_item\_out.append(child.text(1))

                    #self.checked\_item0\_out.append(child.text(0))

        recurse(*self*.tw\_devolucao.invisibleRootItem())

*self*.question\_devolucao('devolucao')

    def question\_devolucao(*self*, *table*):

        msgBox = QMessageBox()

        if *table* == 'não devolucão':

            pass

        else:

            msgBox.setText("Deseja realizar a Devolução?")

            msgBox.setInformativeText("Os livros selecionados voltarão a estar disponíveis \n Clique em Yes para Confirmar")

            msgBox.setStandardButtons(QMessageBox.Yes | QMessageBox.No)

            msgBox.setDetailedText(f"livros: {*self*.checked\_item\_out}")

        msgBox.setIcon(QMessageBox.Question)

        ret = msgBox.exec()

        if ret == QMessageBox.Yes:

            if *table* == "não devolução":

                pass

            else:

                data\_devolucao = date.today()

                data\_devolucao = data\_devolucao.strftime('%d/%m/%y')

*self*.db = DataBase()

*self*.db.conecta()

                #self.relat\_locado('devolucao')

*self*.relat\_locado()

*self*.db.update\_disp\_livro\_devolucao(*self*.checked\_item\_out)

*self*.db.update\_disp\_livro\_lo\_devolucao(data\_devolucao, *self*.checked\_item\_out)

*self*.db.close\_connection()

*self*.reset\_table()

    def relat\_locado(*self*):

        cn = sqlite3.connect('system.db')

        result = pd.read\_sql\_query("SELECT \* FROM locacao WHERE status\_livro\_lo = 'Locado'",  cn)

        result = result.values.tolist()

*self*.x = ""

        for i in result:

*self*.campo = QTreeWidgetItem(*self*.tb\_relatorio, i)

*self*.campo.setCheckState(0, QtCore.Qt.*CheckState*.Unchecked)

*self*.x = i[0]

*self*.tb\_relatorio.setSortingEnabled(True)

        for i in range(0,6):

*self*.tb\_relatorio.resizeColumnToContents(i)

##################################################################################################################################

if \_\_name\_\_ == '\_\_main\_\_':

    app = QApplication(sys.argv)

    window = Login()

    window.show()

    app.exec()

1. **CÓDIGO: DATABASE.PAY**

import sqlite3

class DataBase():

    def \_\_init\_\_(*self*, *name* = "system.db") -> None:

*self*.name = *name*

    def conecta(*self*):

*self*.connection = sqlite3.connect(*self*.name)

    def close\_connection(*self*):

        try:

*self*.connection.close()

        except AttributeError:

            pass

##################################################################################################################################

###\_Usuário\_#######################################################################################################################

    def create\_table\_users(*self*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS users(

                   id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

                   name TEXT NOT NULL,

                   user TEXT UNIQUE NOT NULL,

                   password TEXT NOT NULL,

                   access TEXT NOT NULL,

                   email TEXT NOT NULL,

                   tel TEXT NOT NULL

                );

            """)

        except AttributeError:

            print ("faça a conexão")

    def inser\_user(*self*, *name*, *user*, *password*, *access*, *email*, *tel*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""

                INSERT INTO users(name, user, password, access, email, tel)  VALUES(?,?,?,?,?,?)

            """, (*name*, *user*, *password*, *access*, *email*, *tel*))

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão")

    def check\_user(*self*, *user*, *password*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""

                SELECT \* FROM users;

            """)

            for linha in cursor.fetchall():

                if linha[2].upper() == *user*.upper() and linha[3] == *password* and linha[4] == "Administrador":

                    return "administrador"

                elif linha[2].upper() == *user*.upper() and linha[3] == *password* and linha[4] == "Usuário":

                    return "user"

                else:

                    continue

                return "Sem Acesso"

        except AttributeError:

            pass

    def busca\_user(*self*, *name*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""

                SELECT \* FROM users;

            """)

            for linha in cursor.fetchall():

                if linha[1].upper() == *name*.upper():

                    return  linha[1], linha[2], linha[3], linha[5], linha[6], linha[0]

        except AttributeError:

            pass

    def update\_user(*self*, *fulDataSet*):

        cursor = *self*.connection.cursor()

        cursor.execute(f""" UPDATE users SET

                   name = '{*fulDataSet*[1]}',

                   user = '{*fulDataSet*[2]}',

                   password = '{*fulDataSet*[3]}',

                   email = '{*fulDataSet*[4]}',

                   tel = '{*fulDataSet*[5]}'

                   WHERE id = '{*fulDataSet*[0]}'""")

*self*.connection.commit()

###################################################################################################################################

###\_Livros\_########################################################################################################################

    def create\_table\_livros(*self*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS livros(

                   id\_livro INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

                   nome\_livro TEXT NOT NULL,

                   genero\_livro TEXT NOT NULL,

                   quantidade\_livro INTEGER NOT NULL,

                   autor\_livro TEXT NOT NULL,

                   descricao\_livro TEXT NOT NULL,

                   status\_livro TEXT NOT NULL

                );

            """)

        except AttributeError:

            print ("faça a conexão")

    def select\_all\_livros(*self*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("SELECT \* FROM livros ORDER BY nome\_livro")

            livros = cursor.fetchall()

            return livros

        except AttributeError:

            pass

    def delete\_livros(*self*, *id\_livro*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute(f"DELETE FROM livros WHERE id\_livro = '{*id\_livro*}'")

*self*.connection.commit()

            return "Livro excluido com sucesso!"

        except AttributeError:

            return "Erro ao excluir livro!"

    def inser\_livro(*self*, *fullDataSet*):

        campos\_tabela = ('nome\_livro', 'genero\_livro',  'autor\_livro',  'descricao\_livro', 'status\_livro', 'quantidade\_livro' )

        qntd = ("?,?,?,?,?,?")

        cursor = *self*.connection.cursor()

        try:

            cursor.execute(f"""

                INSERT INTO livros {campos\_tabela}

                VALUES({qntd})""", *fullDataSet*)

*self*.connection.commit()

            return("ok")

        except AttributeError:

            return "erro"

    def update\_livro(*self*, *fulDataSet*):

        cursor = *self*.connection.cursor()

        cursor.execute(f""" UPDATE livros set

                   id\_livro = '{*fulDataSet*[0]}',

                   nome\_livro = '{*fulDataSet*[1]}',

                   genero\_livro = '{*fulDataSet*[2]}',

                   quantidade\_livro = '{*fulDataSet*[3]}',

                   autor\_livro = '{*fulDataSet*[4]}',

                   descricao\_livro = '{*fulDataSet*[5]}',

                   status\_livro = '{*fulDataSet*[6]}'

                   WHERE id\_livro = '{*fulDataSet*[0]}'""")

*self*.connection.commit()

    def update\_disp\_livro(*self*, *id\_li*):

        try:

            cursor = *self*.connection.cursor()

            for idli in *id\_li*:

                cursor.execute(f"""UPDATE livros SET status\_livro = 'Reservado' WHERE nome\_livro = '{idli}'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

    def update\_disp\_livro\_devolucao(*self*, *id\_li*):

        try:

            cursor = *self*.connection.cursor()

            for idli in *id\_li*:

                cursor.execute(f"""UPDATE livros SET status\_livro = 'Disponível' WHERE nome\_livro = '{idli}'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

    def update\_disp\_livro\_locacao(*self*, *data\_locacao*, *data\_prevista\_devolucao*, *usuario*, *id\_li*):

        try:

            cursor = *self*.connection.cursor()

            for idli in *id\_li*:

                cursor.execute(f"""UPDATE locacao SET data\_locacao\_lo = '{*data\_locacao*}', data\_prevista\_devolucao\_lo = '{*data\_prevista\_devolucao*}', usuario\_lo ='{*usuario*}', status\_livro\_lo = 'Reservado' WHERE nome\_livro\_lo = '{idli}'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

    def update\_disp\_livro\_lo\_devolucao(*self*,  *data\_devolucao*,  *id\_li*):

        try:

            cursor = *self*.connection.cursor()

            for idli in *id\_li*:

                cursor.execute(f"""UPDATE locacao SET data\_devolucao\_lo = '{*data\_devolucao*}', status\_livro\_lo = 'Disponível' WHERE nome\_livro\_lo = '{idli}'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

    def estorno\_disp\_livro(*self*, *id\_li*):

        try:

            cursor = *self*.connection.cursor()

            for idli in *id\_li*:

                cursor.execute(f"""UPDATE livros SET status\_livro = 'Disponível' WHERE nome\_livro = '{idli}'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

    def estorno\_disp\_livro\_locacao(*self*, *id\_li*):

        try:

            cursor = *self*.connection.cursor()

            for idli in *id\_li*:

                cursor.execute(f"""UPDATE locacao SET status\_livro\_lo = 'Disponível' WHERE nome\_livro\_lo = '{idli}'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

#####\_Locação\_#####################################################################################################################

    def create\_table\_locacao (*self*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS locacao(

                   id\_livro\_lo INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,

                   nome\_livro\_lo TEXT NOT NULL,

                   quantidade\_livro\_lo INTEGER NOT NULL,

                   data\_locacao\_lo DATE NOT NULL,

                   data\_prevista\_devolucao\_lo DATE NOT NULL,

                   data\_devolucao\_lo DATE NOT NULL,

                   usuario\_lo TEXT NOT NULL,

                   status\_livro\_lo TEXT NOT NULL

                );

            """)

        except AttributeError:

            print ("faça a conexão")

    def inser\_livro\_locacao(*self*, *fullDataSet*):

        campos\_tabela = ('nome\_livro\_lo', 'status\_livro\_lo', 'quantidade\_livro\_lo', 'data\_locacao\_lo', 'data\_prevista\_devolucao\_lo',  'data\_devolucao\_lo', 'usuario\_lo'  )

        qntd = ("?,?,?,?,?,?,?")

        cursor = *self*.connection.cursor()

        try:

            cursor.execute(f"""

                INSERT INTO locacao {campos\_tabela}

                VALUES({qntd})""", *fullDataSet*)

*self*.connection.commit()

            return("ok")

        except AttributeError:

            return "erro"

    def status\_limpo(*self*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""UPDATE livros SET status\_livro = 'Disponível' WHERE status\_livro = 'Reservado'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""UPDATE locacao SET status\_livro\_lo = 'Disponível' WHERE status\_livro\_lo = 'Reservado'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

    def confirmar\_locacao(*self*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""UPDATE livros SET status\_livro = 'Locado' WHERE status\_livro = 'Reservado'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("""UPDATE locacao SET status\_livro\_lo = 'Locado' WHERE status\_livro\_lo = 'Reservado'""")

*self*.connection.commit()

        except AttributeError:

            print("faça a conexão para alterar os campos")

    def busca\_locacao(*self*, *nome\_livro*):

        try:

            cursor = *self*.connection.cursor()

            cursor.execute("SELECT \* FROM locacao;")

            for linha in cursor.fetchall():

                print (linha[1])

                print(*nome\_livro*)

                if linha[1].upper() == *nome\_livro*.upper():

                    return  linha[1], linha[2], linha[3], linha[4], linha[5], linha[6], linha[0]

        except AttributeError:

            pass

        except AttributeError:

            print("faça a conexão para alterar os campos")

###################################################################################################################################

if \_\_name\_\_ == "\_\_main\_\_":

    db = DataBase()

    db.conecta()

    db.create\_table\_users()

    db.create\_table\_livros()

    db.create\_table\_locacao()

    db.close\_connection()