Министерство образования Республики Беларусь  
Учреждение образования  
«Брестский государственный технический университет»  
Кафедра ИИТ

Лабораторная работа №6  
По дисциплине: «Создание приложения для работы с БД организация пользовательского интерфейса: табличное представление и отчеты»

Выполнил:  
студент 3 курса  
группы ИИ-23  
Романюк А. П.

Проверил:  
Савонюк В. А.

Брест 2024

**Цель работы**: получить навыки разработки приложений БД и организации пользовательского интерфейса: табличного представления информации и построения отчетов.

**Задание**

Изучить материал, приведенный в “К лабораторной работе 6.doc”.

На основании логической модели (в соответствии с вариантом Лабораторной работы №1) создать приложение и организовать пользовательский интерфейс, используя соответствующие элементы и методы: организовать табличное представление информации и построить отчеты не менее 4.

**Код программы:**

import tkinter as tk  
from tkinter import ttk  
import sqlite3  
from tkinter import messagebox  
from tkinter import filedialog  
  
class DatabaseApp:  
 def \_\_init\_\_(self, master, connection\_params):  
 self.master = master  
 self.connection\_params = connection\_params  
 self.master.title("СМОТРИ БД")  
  
 self.notebook = ttk.Notebook(master)  
 self.notebook.pack(expand=True, fill='both')  
  
 # Connect to the database  
 self.conn = sqlite3.connect(\*\*connection\_params)  
 self.cursor = self.conn.cursor()  
  
 # Fetch table names  
 self.table\_names = self.get\_table\_names()  
  
 # Create a tab for each table  
 for table\_name in self.table\_names:  
 frame = tk.Frame(self.notebook)  
 self.notebook.add(frame, text=table\_name)  
 self.create\_table\_view(frame, table\_name)  
  
 def get\_table\_names(self):  
 # Fetch table names from the database  
 self.cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';")  
 table\_names = [row[0] for row in self.cursor.fetchall()]  
 return table\_names  
  
 def create\_table\_view(self, frame, table\_name):  
 # Fetch column names  
 self.cursor.execute(f"PRAGMA table\_info({table\_name});")  
 columns = [row[1] for row in self.cursor.fetchall()]  
  
 # Create a treeview widget  
 tree = ttk.Treeview(frame, columns=columns, show='headings', selectmode='browse')  
 tree.pack(expand=True, fill='both')  
  
 # Add column headings with sorting functionality  
 for col in columns:  
 tree.heading(col, text=col, command=lambda c=col: self.sort\_treeview(tree, table\_name, c, False))  
 tree.column(col, width=100, anchor='center')  
  
 # Populate treeview with data from the table  
 self.populate\_treeview(tree, table\_name)  
  
 # Add buttons for CRUD operations  
 add\_button = tk.Button(frame, text="Добавить", command=lambda: self.add\_row(tree, table\_name))  
 add\_button.pack(side=tk.LEFT, padx=10)  
  
 delete\_button = tk.Button(frame, text="Удалить", command=lambda: self.delete\_row(tree, table\_name))  
 delete\_button.pack(side=tk.LEFT, padx=10)  
  
 edit\_button = tk.Button(frame, text="Изменить", command=lambda: self.edit\_row(tree, table\_name))  
 edit\_button.pack(side=tk.LEFT, padx=10)  
  
 refresh\_button = tk.Button(frame, text="Обновить", command=lambda: self.populate\_treeview(tree, table\_name))  
 refresh\_button.pack(side=tk.LEFT, padx=10)  
  
 # Add search functionality  
 search\_entry = tk.Entry(frame)  
 search\_entry.pack(side=tk.LEFT, padx=10)  
  
 search\_button = tk.Button(frame, text="Поиск", command=lambda: self.search\_treeview(tree, search\_entry.get()))  
 search\_button.pack(side=tk.LEFT, padx=10)  
  
 # Add report button  
 report\_button = tk.Button(frame, text="Создать отчет", command=self.generate\_report)  
 report\_button.pack(side=tk.LEFT, padx=10)  
  
 def populate\_treeview(self, tree, table\_name):  
 # Fetch data from the table  
 self.cursor.execute(f"SELECT \* FROM {table\_name};")  
 data = self.cursor.fetchall()  
  
 # Clear existing data in treeview  
 tree.delete(\*tree.get\_children())  
  
 # Insert data into treeview  
 for row in data:  
 tree.insert('', 'end', values=row)  
  
 def add\_row(self, tree, table\_name):  
 # Get column names  
 self.cursor.execute(f"PRAGMA table\_info({table\_name});")  
 columns = [row[1] for row in self.cursor.fetchall()]  
  
 # Create a dialog for adding a new row  
 add\_dialog = tk.Toplevel(self.master)  
 add\_dialog.title("Добавить строку")  
  
 # Entry widgets for each column  
 entry\_widgets = []  
 for col in columns:  
 label = tk.Label(add\_dialog, text=col)  
 label.grid(row=columns.index(col), column=0, padx=10, pady=5, sticky='e')  
 entry = tk.Entry(add\_dialog)  
 entry.grid(row=columns.index(col), column=1, padx=10, pady=5, sticky='w')  
 entry\_widgets.append(entry)  
  
 # Function to insert the new row into the table  
 def insert\_row():  
 values = [entry.get() for entry in entry\_widgets]  
 placeholders = ', '.join(['?' for \_ in values])  
 query = f"INSERT INTO {table\_name} VALUES ({placeholders});"  
 self.cursor.execute(query, values)  
 self.conn.commit()  
 self.populate\_treeview(tree, table\_name)  
 add\_dialog.destroy()  
  
 # Button to submit the new row  
 submit\_button = tk.Button(add\_dialog, text="Подтвердить", command=insert\_row)  
 submit\_button.grid(row=len(columns), columnspan=2, pady=10)  
  
 def delete\_row(self, tree, table\_name):  
 # Get the selected item in the treeview  
 selected\_item = tree.selection()  
 if not selected\_item:  
 messagebox.showwarning("Предупреждение", "Пожалуйста, выберите строку для удаления.")  
 return  
  
 # Confirm deletion  
 confirm = messagebox.askyesno("Подтверждение", "Вы уверены, что хотите удалить эту строку?")  
 if not confirm:  
 return  
  
 # Get the values of the selected row  
 values = tree.item(selected\_item)['values']  
  
 # Create a WHERE clause for deletion  
 where\_clause = ' AND '.join([f"{column} = ?" for column in tree['columns']])  
  
 # Execute the DELETE query  
 query = f"DELETE FROM {table\_name} WHERE {where\_clause};"  
 self.cursor.execute(query, values)  
 self.conn.commit()  
  
 # Update the treeview  
 self.populate\_treeview(tree, table\_name)  
  
 def edit\_row(self, tree, table\_name):  
 # Get the selected item in the treeview  
 selected\_item = tree.selection()  
 if not selected\_item:  
 messagebox.showwarning("Предупреждение", "Пожалуйста, выберите строку для изменения.")  
 return  
  
 # Get the values of the selected row  
 values = tree.item(selected\_item)['values']  
  
 # Get column names  
 self.cursor.execute(f"PRAGMA table\_info({table\_name});")  
 columns = [row[1] for row in self.cursor.fetchall()]  
  
 # Create a dialog for editing the row  
 edit\_dialog = tk.Toplevel(self.master)  
 edit\_dialog.title("Изменить строку")  
  
 # Entry widgets for each column with current values  
 entry\_widgets = []  
 for col, value in zip(columns, values):  
 label = tk.Label(edit\_dialog, text=col)  
 label.grid(row=columns.index(col), column=0, padx=10, pady=5, sticky='e')  
 entry = tk.Entry(edit\_dialog)  
 entry.insert(0, value)  
 entry.grid(row=columns.index(col), column=1, padx=10, pady=5, sticky='w')  
 entry\_widgets.append(entry)  
  
 # Function to update the row in the table  
 def update\_row():  
 new\_values = [entry.get() for entry in entry\_widgets]  
 set\_clause = ', '.join([f"{column} = ?" for column in columns])  
 where\_clause = ' AND '.join([f"{column} = ?" for column in columns])  
 query = f"UPDATE {table\_name} SET {set\_clause} WHERE {where\_clause};"  
 self.cursor.execute(query, new\_values + values)  
 self.conn.commit()  
 self.populate\_treeview(tree, table\_name)  
 edit\_dialog.destroy()  
  
 # Button to submit the edited row  
 submit\_button = tk.Button(edit\_dialog, text="Подтвердить", command=update\_row)  
 submit\_button.grid(row=len(columns), columnspan=2, pady=10)  
  
 def sort\_treeview(self, tree, table\_name, column, reverse):  
 # Fetch data from the table with sorting  
 query = f"SELECT \* FROM {table\_name} ORDER BY {column} {'DESC' if reverse else 'ASC'};"  
 self.cursor.execute(query)  
 data = self.cursor.fetchall()  
  
 # Clear existing data in treeview  
 tree.delete(\*tree.get\_children())  
  
 # Insert sorted data into treeview  
 for row in data:  
 tree.insert('', 'end', values=row)  
  
 tree.heading(column, command=lambda: self.sort\_treeview(tree, table\_name, column, not reverse))  
  
  
 def search\_treeview(self, tree, search\_term):  
 # Iterate over each row in the treeview  
 for item in tree.get\_children():  
 values = tree.item(item)['values']  
  
 # Check if the search term is present in any of the values  
 if any(str(search\_term).lower() in str(value).lower() for value in values):  
 tree.selection\_add(item)  
 else:  
 tree.selection\_remove(item)  
  
 def generate\_report(self):  
 # Create a file dialog to choose where to save the report  
 file\_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text files", "\*.txt")])  
  
 if not file\_path:  
 return # User canceled the operation  
  
 # Create or open the file for writing  
 with open(file\_path, 'w') as report\_file:  
 # Write information about each table to the report file  
 for table\_name in self.table\_names:  
 report\_file.write(f"Таблица: {table\_name}\n")  
  
 # Fetch column names  
 self.cursor.execute(f"PRAGMA table\_info({table\_name});")  
 columns = [row[1] for row in self.cursor.fetchall()]  
  
 # Write column names to the report file  
 report\_file.write("\tСтолбцы: " + ", ".join(columns) + "\n")  
  
 # Fetch and write data to the report file  
 self.cursor.execute(f"SELECT \* FROM {table\_name};")  
 data = self.cursor.fetchall()  
 report\_file.write("\tДанные:\n")  
 for row in data:  
 report\_file.write("\t\t" + ", ".join(str(value) for value in row) + "\n")  
  
 report\_file.write("\n")  
  
 # Fetch row count  
 self.cursor.execute(f"SELECT COUNT(\*) FROM {table\_name};")  
 row\_count = self.cursor.fetchone()[0]  
 report\_file.write(f"Количество строк: {row\_count}\n\n")  
  
 messagebox.showinfo("Создание отчета", "Отчет успешно создан.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 connection\_params = {"database": "mydb.sqlite3"}  
 try:  
 root = tk.Tk()  
 app = DatabaseApp(root, connection\_params)  
 root.mainloop()  
  
 except sqlite3.Error as err:  
 print(f"Error: {err}")

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

**Вывод**: в ходе лабораторной работы я создал приложение с графическим интерфейсом для управления базой данных.