

Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет информатики и
радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа № 6
«Создание приложения для базы данных»
Вариант № 21 (Оператор связи)

Выполнил
студент группы 150503:
Тюшкевич М.А.

Проверила:
Игнатович А.О.

Минск 2024

1 Цель работы

В ходе выполнения лабораторной работы необходимо разработать приложение для работы с базой данных.

2 Выполнение работы

Для разработки приложения был выбран язык Python, который позволяет быстро реализовать требуемый функционал.

В качестве библиотеки для графического интерфейса была выбрана библиотека Tkinter. В этой библиотеке есть большой выбор различных графических элементов с разнообразными настройками.

Для взаимодействия с базой данных был выбран модуль Psycopg2. Из его преимуществ можно выделить подробную документацию, распространенность и поддержку многопоточности.

```
self.conn = psycopg2.connect(dbname='Telecom',
user='postgres', host='localhost')
self.cursor = self.conn.cursor()
```

Метод `connect` принимает в качестве аргументов параметры базы данных, далее получается объект `cursor`, который используется для манипуляций с базой данных.

```
self.cursor.execute(data)
```

Метод `execute` получает обычный SQL-запрос и отправляет его в базу данных. Данные берутся из поля в GUI, данные вводит пользователь.

```
result = self.cursor.fetchall()
```

Метод `fetchall` у объекта `curcor` получает все строки, которые вернула база данных.

Методы выше реализуют получение информации из базы данных. Далее показан процесс добавления новых данных в базу.

```
self.cursor.execute(data)
self.conn.commit()
```

Метод `commit` подтверждает любые манипуляции с БД, без него никакие действия не сохранились бы.

На рисунке 2.1 показан интерфейс программы. Есть две строки ввода. Первая строка позволяет выполнить SELECT-запрос, а вторая INSERT-запрос.

The screenshot shows a window titled "Telecom Application". It contains two main input sections: "Query:" and "Insert:". Each section has a text input field and a corresponding button ("Query data" and "Insert data" respectively). Below these are several control buttons: "Select table" (with a dropdown arrow), "Add Row", "Modify Row", "Delete Row", "Column to sort" (with a dropdown arrow), and "Sort type" (with a dropdown arrow). The central area is a large empty rectangle intended for a table. At the bottom, there is an "Error info:" label followed by a text area.

Рисунок 2.1 – Интерфейс приложения

С помощью комбобокса можно выбрать таблицу для просмотра. Будут выведены все поля в таблице и названия колонок.

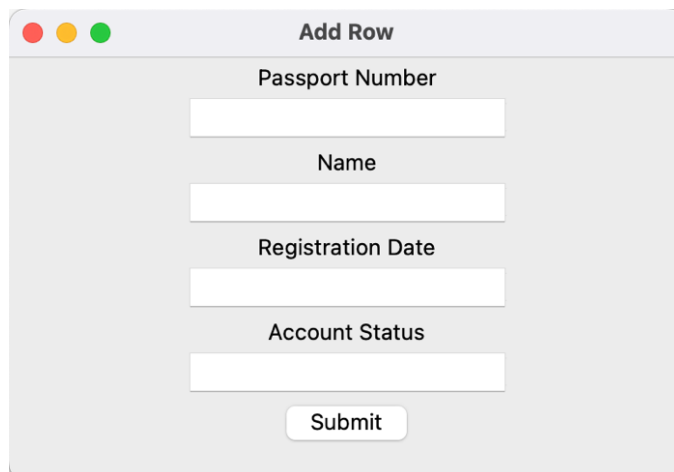
This screenshot shows the same "Telecom Application" window, but now it displays a table of data. The "Query:" field contains the text "SELECT * FROM telecom.balances". The "Select table" dropdown is set to "balances". The table has four columns: "id", "amount", "change_date", and "phone_number". The data is as follows:

id	amount	change_date	phone_number
aaa90162-c73f-4821-8d4d-e8;	-\$10.00	2023-01-20 00:00:00	+375 44 134 24 35
2fdb26c-6289-40d3-86d2-4c	\$75.25	2023-02-25 00:00:00	+375 29 488 35 33
1b4deec0-8fd4-47a6-bd26-4d	\$93.25	2023-02-25 00:00:00	+375 33 256 25 76
196fac45-5f6a-4880-8bf1-791	\$14.50	2023-04-10 00:00:00	+375 33 930 67 50
4b2be110-e0d7-49b5-9ac4-b1	\$92.70	2023-05-17 00:00:00	+375 44 144 24 23
d2c36850-6956-4ae3-a7ff-fd	\$220.00	2023-01-20 00:00:00	+375 33 175 11 14
82d66034-5914-497f-b4c1-13	\$163.00	2023-03-15 00:00:00	+375 17 159 59 94
5661b5c3-e44c-4ee4-a5f0-e5	\$11.00	2023-03-15 00:00:00	+375 29 996 73 04
e5d5440d-4e5d-4d43-887e-c	\$77.00	2023-04-10 00:00:00	+375 44 119 20 35

The "Error info:" field at the bottom is empty.

Рисунок 2.2 – Пример вывода запроса

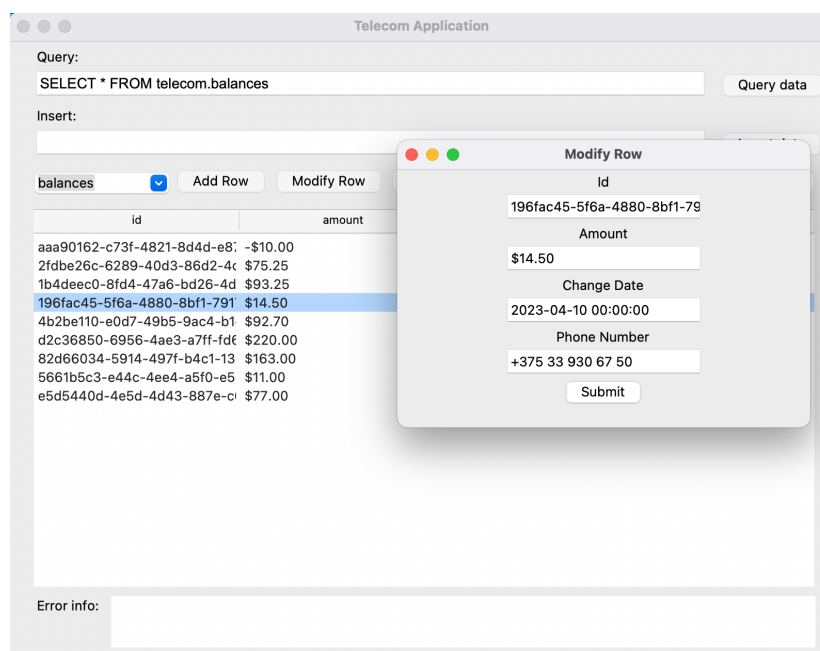
При нажатии на кнопку «Add row» появится окно, с полями для ввода информации. Если в поле ничего не указать, то будет использовано значение по умолчанию. В случае неверного формата входных данных описание ошибки будет выведено в поле «Error info» в главном окне.



The 'Add Row' dialog box is a simple form with a light gray background. It has a title bar with three colored buttons (red, yellow, green) on the left. The form contains four text input fields, each with a label above it: 'Passport Number', 'Name', 'Registration Date', and 'Account Status'. Below the last field is a 'Submit' button.

Рисунок 2.3 – Окно добавления новой записи в таблицу

Окно редактирования строк аналогично окну добавления, но поля уже заполнены, и пользователь может изменить что-то по своему желанию. Пример редактирования на рисунке 2.4.



The 'Telecom Application' window displays a table with two columns: 'id' and 'amount'. The table contains several rows of data. A row with id '196fac45-5f6a-4880-8bf1-791' and amount '\$14.50' is selected. A 'Modify Row' dialog box is open over the table, showing the same row's data in input fields: 'Id' (196fac45-5f6a-4880-8bf1-791), 'Amount' (\$14.50), 'Change Date' (2023-04-10 00:00:00), and 'Phone Number' (+375 33 930 67 50). The dialog has a 'Submit' button. The main window also has a 'Query' field with the text 'SELECT * FROM telecom.balances', an 'Insert' field, and buttons for 'Add Row' and 'Modify Row'. An 'Error info' field is at the bottom.

id	amount
aaa90162-c73f-4821-8d4d-e8f1	\$10.00
2fdbe26c-6289-40d3-86d2-4c1b	\$75.25
1b4deec0-8fd4-47a6-bd26-4d1a	\$93.25
196fac45-5f6a-4880-8bf1-791	\$14.50
4b2be110-e0d7-49b5-9ac4-b1d2	\$92.70
d2c36850-6956-4ae3-a7ff-fd6e	\$220.00
82d66034-5914-497f-b4c1-13e5	\$163.00
5661b5c3-e44c-4ee4-a5f0-e5e5	\$11.00
e5d5440d-4e5d-4d43-887e-c1a1	\$77.00

Рисунок 2.4 – Окно редактирования записи в таблице

Для удаления существует кнопка «Delete row» для ее использования необходимо выделить строку для удаления и нажать кнопку.

Так же есть два комбобокса, которые позволяют выбрать колонку в текущей таблице и выбрать, как ее отсортировать. Пример сортировки на рисунке 2.5.

Telecom Application

Query:

Insert:

balances amount

id	amount	change_date	phone_number
aaa90162-c73f-4821-8d4d-e87	-\$10.00	2023-01-20 00:00:00	+375 44 134 24 35
5661b5c3-e44c-4ee4-a5f0-e5	\$11.00	2023-03-15 00:00:00	+375 29 996 73 04
196fac45-5f6a-4880-8bf1-791	\$14.50	2023-04-10 00:00:00	+375 33 930 67 50
2fdbe26c-6289-40d3-86d2-4c	\$75.25	2023-02-25 00:00:00	+375 29 488 35 33
e5d5440d-4e5d-4d43-887e-c	\$77.00	2023-04-10 00:00:00	+375 44 119 20 35
4b2be110-e0d7-49b5-9ac4-b1	\$92.70	2023-05-17 00:00:00	+375 44 144 24 23
1b4deec0-8fd4-47a6-bd26-4d	\$93.25	2023-02-25 00:00:00	+375 33 256 25 76
82d66034-5914-497f-b4c1-13	\$163.00	2023-03-15 00:00:00	+375 17 159 59 94
d2c36850-6956-4ae3-a7ff-fd	\$220.00	2023-01-20 00:00:00	+375 33 175 11 14

Error info:

Рисунок 2.5 – Пример сортировки записей в таблице

Insert data

amount

change_date

00:00:00 +375 17 159 59 94

00:00:00 +375 29 996 73 04

Рисунок 2.6– Способы сортировки записей в таблице

3 Листинг кода

```
import tkinter as tk
from tkinter import ttk
import psycopg2
```

```

class TelecomApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Telecom Application")
        self.root.geometry('800x600')

        self.label1 = self.create_label('Query:', 0.03, 0.005)
        self.entry1 = self.create_input_entry(root, 80, 0.03,
0.045, self.get_data, '<Return>')

        self.label2 = self.create_label('Insert:', 0.03, 0.1)
        self.entry2 = self.create_input_entry(root, 80, 0.03, 0.14,
self.pull_data, '<Return>')

        self.get_button = tk.Button(root, text='Query data',
command=self.get_data)
        self.get_button.place(relx=0.86, rely=0.045)

        self.pull_button = tk.Button(root, text='Insert data',
command=self.pull_data)
        self.pull_button.place(relx=0.86, rely=0.14)

        self.add_button = tk.Button(root, text='Add Row',
command=self.add_row)
        self.add_button.place(relx=0.42, rely=0.2)

        self.delete_button = tk.Button(root, text='Delete Row',
command=self.delete_row)
        self.delete_button.place(relx=0.68, rely=0.2)

        self.modify_button = tk.Button(root, text='Modify Row',
command=self.modify_row)
        self.modify_button.place(relx=0.54, rely=0.2)

        self.table_combobox = ttk.Combobox(root, state="readonly",
width=30)
        self.table_combobox.place(relx=0.03, rely=0.21)

        self.tree = ttk.Treeview(root, show="headings")
        self.tree.place(relx=0.03, rely=0.27, relwidth=0.945,
relheight=0.61)

        self.info_label = self.create_label('Error info:', 0.03,
0.89)
        self.info_text = self.create_text_widget(root, 85, 3, 0.12,
0.89)

        self.conn = psycopg2.connect(dbname='Telecom',
user='postgres', host='localhost')
        self.cursor = self.conn.cursor()

        self.populate_table_combobox()
        self.table_combobox.bind("<<ComboboxSelected>>",
self.display_selected_table)

```

```

    def create_input_entry(self, parent, width, relx, rely,
bind_function, action='<Return>'):
        input_entry = tk.Entry(parent, width=width, font=('Arial',
14))
        input_entry.place(relx=relx, rely=rely)
        input_entry.bind(action, lambda event: bind_function())
        return input_entry

    def create_label(self, text, relx, rely):
        label = tk.Label(self.root, text=text)
        label.place(relx=relx, rely=rely)
        return label

    def create_text_widget(self, parent, width, height, relx,
rely):
        text_widget = tk.Text(parent, width=width, height=height,
wrap=tk.WORD, font=('Arial', 14), state='disabled')
        text_widget.place(relx=relx, rely=rely)
        return text_widget

    def populate_table_combobox(self):
        try:
            self.cursor.execute("SELECT table_name FROM
information_schema.tables WHERE table_schema='telecom'")
            tables = self.cursor.fetchall()
            table_names = [table[0] for table in tables]
            self.table_combobox['values'] = table_names
        except psycopg2.Error as e:
            self.error_handler(e)

    def display_selected_table(self, event):
        selected_table = self.table_combobox.get()
        query = f"SELECT * FROM telecom.{selected_table}"
        self.entry1.delete(0, tk.END)
        self.entry1.insert(tk.END, query)
        self.get_data()

    def clear_info_text(self):
        self.info_text.config(state='normal')
        self.info_text.delete(1.0, tk.END)
        self.info_text.config(state='disabled')

    def error_handler(self, e):
        self.tree.delete(*self.tree.get_children())
        self.conn.rollback()
        error_message = f"Error executing query: {e}\n"
        self.info_text.config(state='normal')
        self.info_text.insert(tk.END, error_message)
        self.info_text.config(state='disabled')

    def get_data(self):
        self.clear_info_text()
        data = self.entry1.get()
        try:
            self.cursor.execute(data)

```

```

        result = self.cursor.fetchall()

        columns = [desc[0] for desc in self.cursor.description]
        num_columns = len(columns)

        self.tree.configure(columns=tuple(f"column{i}" for i in
range(num_columns)))

        for idx, column in enumerate(columns):
            self.tree.heading(f"column{idx}", text=column)

        self.tree.delete(*self.tree.get_children())

        for row in result:
            self.tree.insert('', tk.END, values=row)

    except psycopg2.Error as e:
        self.error_handler(e)

def pull_data(self):
    self.clear_info_text()
    data = self.entry2.get()
    try:
        self.cursor.execute(data)
        self.conn.commit()

    except psycopg2.Error as e:
        self.error_handler(e)

def add_row(self):
    selected_table = self.table_combobox.get()
    AddRowWindow(self.root, self.conn, selected_table,
self.get_data, self.error_handler)

def delete_row(self):
    selected_table = self.table_combobox.get()
    selected_item = self.tree.selection()
    if selected_item:
        r1 = self.tree.item(selected_item)['values'][0]
        r2 = self.tree.item(selected_item)['values'][1]
        # Assuming first column is row id
        attr1 = self.cursor.description[0][0]
        attr2 = self.cursor.description[1][0]
        try:
            delete_query = f"DELETE FROM
telecom.{selected_table} WHERE {attr1}='{r1}' AND {attr2}='{r2}'"
            self.cursor.execute(delete_query)
            self.conn.commit()
            self.get_data()
        except psycopg2.Error as e:
            self.error_handler(e)

def modify_row(self):
    selected_table = self.table_combobox.get()
    selected_item = self.tree.selection()
    if selected_item:

```



```

        row_values = self.tree.item(selected_item)['values']
        ModifyRowWindow(self.root, self.conn, selected_table,
row_values, self.get_data, self.error_handler)

    def __del__(self):
        self.cursor.close()
        self.conn.close()

class AddRowWindow:
    def __init__(self, parent, conn, table_name, refresh_callback,
error_callback):
        self.parent = parent
        self.conn = conn
        self.table_name = table_name
        self.refresh_callback = refresh_callback
        self.error_callback = error_callback

        self.top = tk.Toplevel(parent)
        self.top.title("Add Row")

        self.entry_values = []
        self.entries = []
        self.populate_entries()

        self.submit_button = tk.Button(self.top, text='Submit',
command=self.submit)
        self.submit_button.pack()

    def populate_entries(self):
        try:
            conn = self.conn
            cursor = conn.cursor()
            cursor.execute(f"SELECT column_name FROM
information_schema.columns WHERE table_name = '{self.table_name}'
ORDER BY ordinal_position")
            columns = cursor.fetchall()

            height = (len(columns) + 1) * 50
            self.top.geometry(f'400x{height}')

            for col in columns:
                label_text = col[0].replace('_', ' ').title()
                label = tk.Label(self.top, text=label_text)
                label.pack()
                entry = tk.Entry(self.top)
                entry.pack()
                self.entry_values.append(col[0])
                self.entries.append(entry)
        except psycopg2.Error as e:
            self.error_callback(e)

    def submit(self):
        values = [entry.get() for entry in self.entries]
        non_empty_values = [value for value in values if
value.strip()]

```

```

        non_empty_entry_values = [column for column, value in
zip(self.entry_values, values) if value.strip()]

        columns = ", ".join(non_empty_entry_values)
        placeholders = ", ".join(['%s' for _ in non_empty_values])
        insert_query = f"INSERT INTO telecom.{self.table_name}
({columns}) VALUES ({placeholders})"
        try:
            cursor = self.conn.cursor()
            cursor.execute(insert_query, non_empty_values)
            self.conn.commit()
            self.refresh_callback()
            self.top.destroy()
        except psycopg2.Error as e:
            self.error_callback(e)

class ModifyRowWindow:
    def __init__(self, parent, conn, table_name, row_values,
refresh_callback, error_callback):
        self.parent = parent
        self.conn = conn
        self.table_name = table_name
        self.row_values = row_values
        self.refresh_callback = refresh_callback
        self.error_callback = error_callback

        self.top = tk.Toplevel(parent)
        self.top.title("Modify Row")

        self.entry_values = []
        self.entries = []
        self.populate_entries()

        self.submit_button = tk.Button(self.top, text='Submit',
command=self.submit)
        self.submit_button.pack()

    def populate_entries(self):
        try:
            conn = self.conn
            cursor = conn.cursor()
            cursor.execute(f"SELECT column_name FROM
information_schema.columns WHERE table_name = '{self.table_name}'
ORDER BY ordinal_position")
            columns = cursor.fetchall()

            height = (len(columns) + 1) * 50
            self.top.geometry(f'400x{height}')

            for idx, col in enumerate(columns):
                label_text = col[0].replace('_', ' ').title()
                label = tk.Label(self.top, text=label_text)
                label.pack()
                entry = tk.Entry(self.top)
                entry.insert(0, self.row_values[idx])

```

```

        entry.pack()
        self.entry_values.append(col[0])
        self.entries.append(entry)
    except psycopg2.Error as e:
        self.error_callback(e)

    def submit(self):
        conn = self.conn
        cursor = conn.cursor()
        cursor.execute(f"SELECT column_name FROM
information_schema.columns WHERE table_name = '{self.table_name}'
ORDER BY ordinal_position")
        columns = cursor.fetchall()

        values = [entry.get() for entry in self.entries]
        update_query = f"UPDATE telecom.{self.table_name} SET "
        update_query += ", ".join([f"{col} = %s" for col in
self.entry_values])

        id1 = columns[0][0]
        id2 = columns[1][0]
        update_query += f" WHERE {id1} = %s AND {id2} = %s"

        try:
            cursor = self.conn.cursor()
            cursor.execute(update_query, values +
[self.row_values[0]] + [self.row_values[1]])
            self.conn.commit()
            self.refresh_callback()
            self.top.destroy()
        except psycopg2.Error as e:
            self.error_callback(e)
            print(e)

if __name__ == "__main__":
    root = tk.Tk()
    app = TelecomApp(root)
    root.mainloop()

```