МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Дальневосточный федеральный университет»
(ДВФУ)

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ (ШКОЛА)

Департамент математического и компьютерного моделирования

«Персональный каталогизатор-систематизатор фотографий анималистического жанра на примере бёрдвотчинга»

КУРСОВАЯ РАБОТА
по образовательной программе подготовки бакалавров
по направлению 02.03.01 «Математика и компьютерные науки»
профиль «Сквозные цифровые технологии»

Студент группы № Б9119-02.03.01сцт

_____ Панченко Н.К
(подпись)

Работа защищена
с оценкой _____

«_____» _____ 2022г.

Руководитель *ст. преподаватель*
(должность, ученое звание)

_____ *Малыкина И.А.*
(подпись) (ФИО)

«*9*» *июня* 2022г.

.

г. Владивосток
2022

Оглавление

Аннотация

В данной работе рассматривается задача реализации персональной базы данных для фотографа-анималиста. Полная реализация базы данных или каталогизатора должны включать следующий функционал:

- создание нового вида,
- загрузка фотографии или серий фотографий,
- привязка к фотографии заметок и геолокации,
- галерея по виду,
- выборку по критериям,
- галерея лучших снимков,
- возможность добавлять видеофрагменты и записи звуков.

В рамках курсовой работы требуется выполнить следующую часть работ: создать приложение для ПК, в котором будут следующие функции: добавление нового вида, общее описание вида, загрузка изображений, галерея по виду.

Решение выполнено с помощью интегрированной среды разработки PyCharm для языка программирования Python.

Введение

Бердвотчинг — это вид активного отдыха очень популярный в Америке и Западной Европе. Заключается в наблюдении за птицами, чаще всего с использованием бинокля или подзорной трубы, или фотографировании птиц, записи их голосов. Значительная часть научного орнитологического материала в мире собирается любителями-бёрдвотчерами. В тех уголках мира, куда бёрдвотчинг пришел в последние 25 лет, например в Азии и на постсоветском пространстве, в большей степени популярен фотобёрдвотчинг, то есть фотографирование птиц. Это связано в первую очередь с тем, что в это время широкой аудитории стали доступны как зеркальные камеры с телеобъективами, так и фотоаппараты-суперзумы. Каждый владелец такого фотоаппарата рано или поздно обязательно начинает делать фотографии птиц. Причин тому несколько. Во-первых, птицы заметны и хороши собой, но люди это замечают только взяв в руки бинокль или фотоаппарат. Во-вторых, птицы разнообразны, и у фотографа появляется желание находить и делать фотографии разных птиц. В-третьих, процесс фотоохоты сопровождается азартом и страстью, то есть, позволяет проявить древние инстинкты, но без крови. И в-четвертых, результат фотобёрдвотчинга — это прекрасные фотографии.

Цель создать персональное приложение для ПК, позволяющее фотографу-анималисту организовать каталог-систематизатор метаданных для личного архива анималистических фотографий. Например, редактируемый справочник видов, полевые заметки, географические привязки, дата съемки. Система должна предоставлять возможности поиска, фильтрации, систематизации фотографий.

Основная часть

Глава 1. Проект

Каталогизатор должен включать следующий функционал: создание нового вида, загрузка фотографии или серий фотографий, привязка к фотографии заметок и геолокации, личные заметки по виду, галерея по виду, выборку по критериям, галерея лучших снимков, возможность добавлять видеофрагменты и записи звуков. Самое главное каталогизатор должен быть персональным.

Приложение	**Персональный фото справо**	**Добавление фотографий**	**Карта**	**Выборка по фильтрам**	**Добавление вида**	**Лучшие снимки**	**Видео аудио**	**Описание вида**	**Не коммерческое**
Мое приложение	+	+	+	+	+	+	+	+	+
Adobe Lightroom	+	+	-	+	-	+	-	-	-
Adobe Bridge	+	+	-	+	-	-	-	-	-
Corel AfterShot Pro	+	-	-	+	-	+	+	-	-
ACDSee Photo Studio	+	-	-	+	-	+	+	-	-
ПДВ	-	+	+	+	+	+	+ -	+	+

ПДВ – сайт птицы Дальнего Востока функционально подходит, но не является персональным.

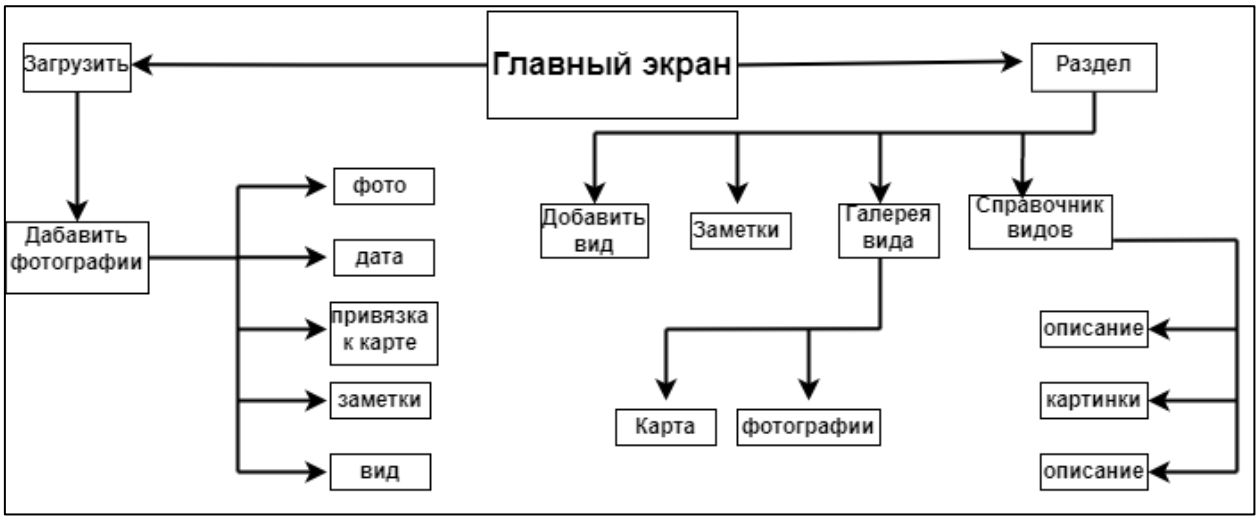

Схема приложения

За основу общей организации интерфейса взят интерфейс интернет-ресурса «Птицы Дальнего Востока», части общего проекта бёрдвотчеров постсоветского пространства. Например:

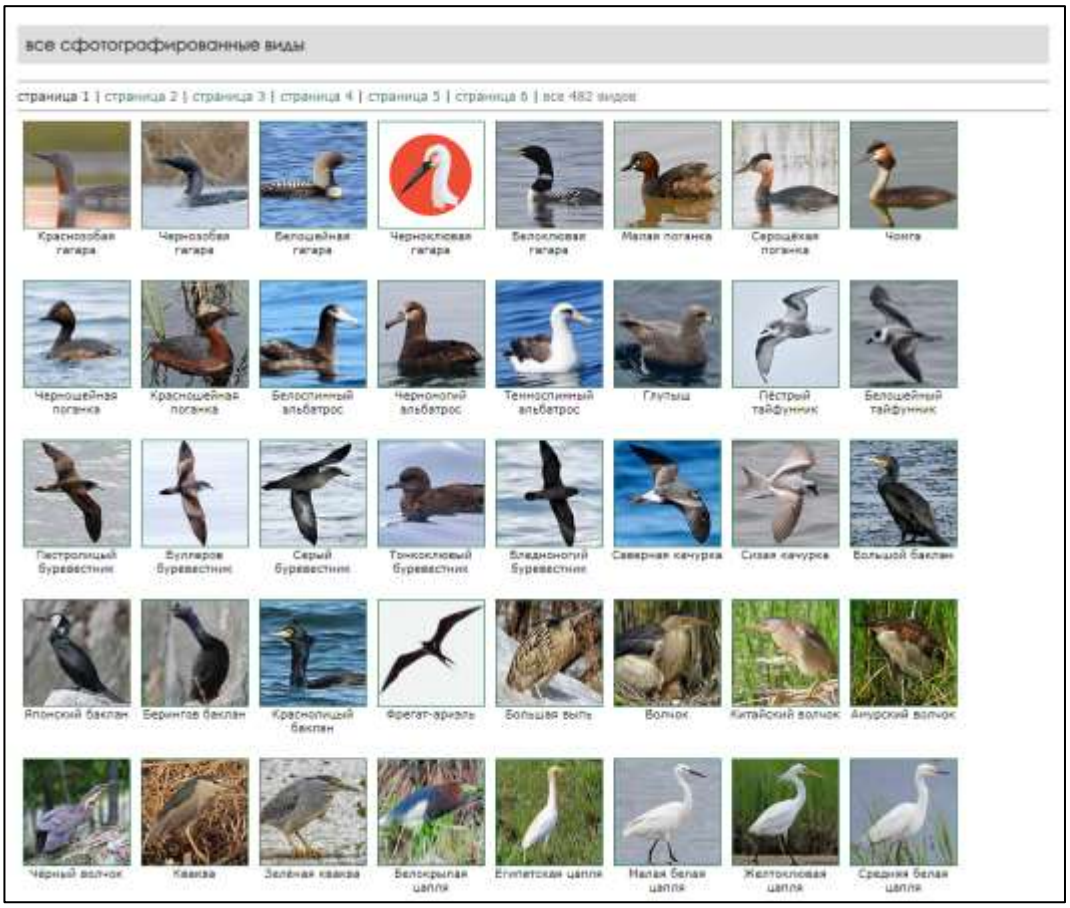

Рисунок 1: страница видов.

Рисунок 2: страница с общей информацией.

Рисунок 3: страница галереи.

Глава 2. Реализация проекта

Требования к окружению

Для разработки использовалась среда разработки PyCharm с использованием языка программирования Python и встроенными библиотеками, и модулями.

Системные требования для установки PyCharm:

- 64 – бит версия Windows 10, 8;
- 8ГБ оперативной памяти;
- 2.5ГБ свободного пространства на диске;
- Python 2.7, или Python 3.5 или новее.

PyCharm

Это кроссплатформенная среда разработки, которая совместима с Windows, macOS, Linux для языка программирования Python. Предоставляет средства для анализа кода, графический отладчик, инструмент для запуска юнит-тестов и поддерживает веб-разработку на Django(программа).

Для работы использовался Python-3.9, библиотеки Tkinter, PIL и модули SQlite3, IO.

Python

Высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами. Стандартные библиотека включает большой объём полезных функций.

SQLite3

Модуль языка Python предоставляет интерфейс SQL, совместимый со спецификацией DB-API 2.0, описанной в PEP 249(спецификация API базы данных Python), и требует SQLite 3.7.15 или новее.

SQL

Декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

SQLite

SQLite — это библиотека C, которая предоставляет легкую дисковую базу данных, не требующую отдельного серверного процесса и позволяющую обращаться к базе данных с помощью нестандартного варианта языка запросов SQL. Некоторые приложения могут использовать SQLite для внутреннего хранения данных.

PIL(Python Imaging Library)

Библиотека языка Python (версии 2), предназначенная для работы с растровой графикой.

Возможности библиотеки:

- поддержка форматов BMP, EPS, GIF, JPEG, PDF, PNG, PNM, TIFF и некоторых других на чтение и запись;
- преобразование изображений из одного формата в другой;
- правка изображений (использование различных фильтров, масштабирование, рисование, матричные операции и т. д.).

IO

Модуль IO для выполнения операций ввода-вывода, связанные с файлами (например, чтение / запись файлов).

Tkinter

Это графическая библиотека, позволяющая создавать программы с оконным интерфейсом.

Проект

Методы и классы

Класс DB создаёт три таблицы для хранения информации с помощью запросов:

```
self.c.execute(
            '''CREATE TABLE IF NOT EXISTS birds (
            id integer primary key, species text,
date_str, date_int integer, place text, pic blob,
notes text)''')

self.c.execute(
            '''CREATE TABLE IF NOT EXISTS species (
            species_id integer primary key, name text,
name_lat text, name_ing text, notes text, places text,
biologi text, pic blob)''')

self.c.execute(
            '''CREATE TABLE IF NOT EXISTS picture (
            id integer primary key, name text, pic
blob, pic_min blob, pic_mid blob)''')
```

Листинг 1

Класс ***Main*** запускает начальное окно программы.

Метод ***tk.Button*** – добавляет четыре кнопки действий(добавить категорию, добавить вид, добавить изображения, удалить). К кнопкам добавляются иконки.

Метод ***ttk.Treeview*** – создается таблица для отображения данных.

Метод ***tk.Scrollbar*** – добавляет полосу прокрутки для таблицы.

```
tk.Button(toolbar, text='Добавить вид', bg='#d7d8e0',
bd=0, image=self.species,
                                    compound=tk.TOP,
command=self.open_dialog_species)
ttk.Treeview(self, columns=('species', 'data'))
tk.Scrollbar(self, command=self.tree.yview)
```

Листинг 2

Метод ***view_records*** – отображает данные в таблице.

```
def view_records(self):
        self.db.c.execute('''SELECT pic, species,
date_str FROM birds''')
        [self.tree.delete(i) for i in
self.tree.get_children()]
        self.imglist = []
        for record in self.db.c.fetchall():
            img = ImageTk.PhotoImage(data=record[0])

            self.tree.insert("", 'end', image=img,
values=record[1:])
            self.imglist.append(img)
```

Листинг 3

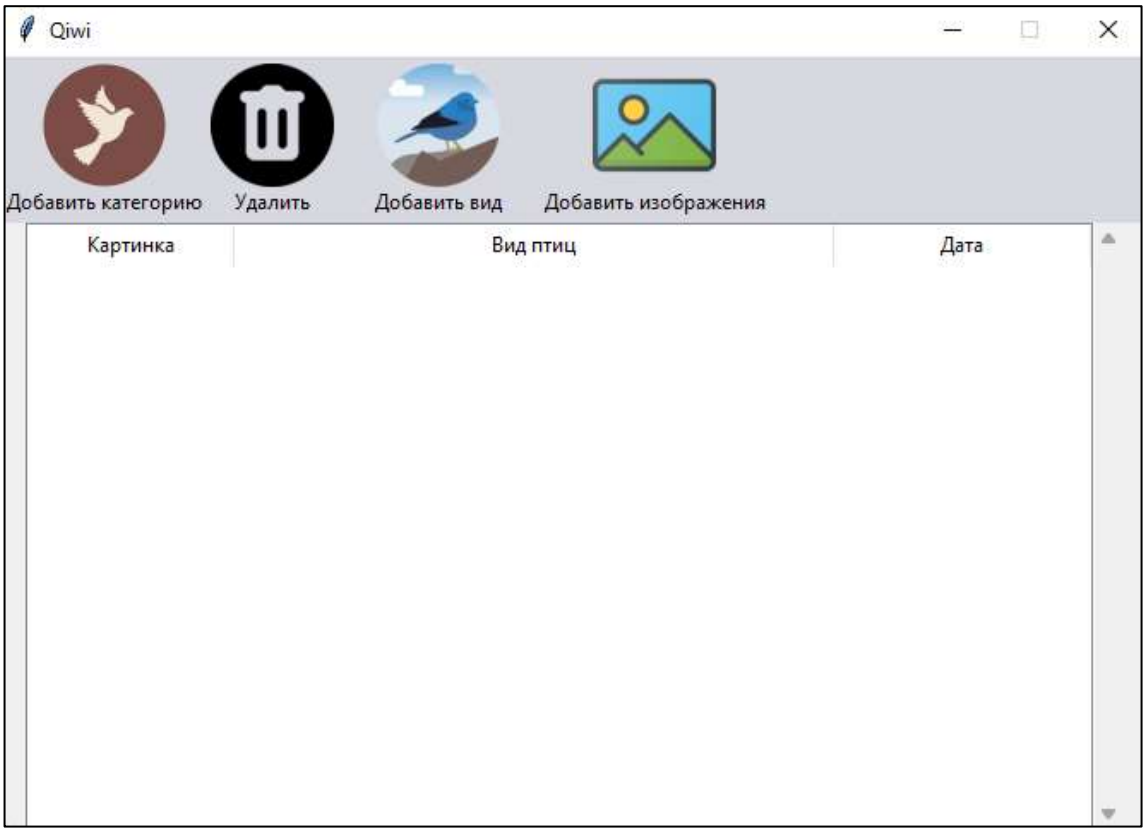

Рисунок 4: основное окно приложения.

Кнопка «Добавить вид» открывает окно, в котором можно создать новый вид для категорий.

Создаются поля для внесения информации методом ***ttk.Entry***.

Для больших полей с описание используется метод ***tk.Text***.

Для добавлений нужно нажать на кнопку «Добавить» вызовется метод, который считает данные с полей и добавит данные в таблицу в которой хранится информация с видами.

```
label_speceis = tk.Label(self, text='Название вида на
русском:')
        label_speceis.place(x=15, y=20)
        self.entry_species = ttk.Entry(self, width=25)
        self.entry_species.place(x=200, y=20)

label_notes = tk.Label(self, text='Описание:')
        label_notes.place(x=15, y=130)
        self.text_notes = tk.Text(self,width=43,
height=8)
        self.text_notes.place(x=13, y=160)

btn_submit = tk.Button(self, text='Добавить',
width=20, font=('Helvetica', 11, 'bold'))
        btn_submit.place(x=110, y=648)
        btn_submit.bind('<Button-1>', lambda event:
self.view.records_for_speces(self.entry_species.get(),

self.entry_species_lat.get(),

self.entry_species_en.get(),

self.text_notes.get(1.0, 'end-1c'),

self.text_places.get(1.0, 'end-1c'),

self.text_biologi.get(1.0, 'end-1c')))
```

Листинг 4

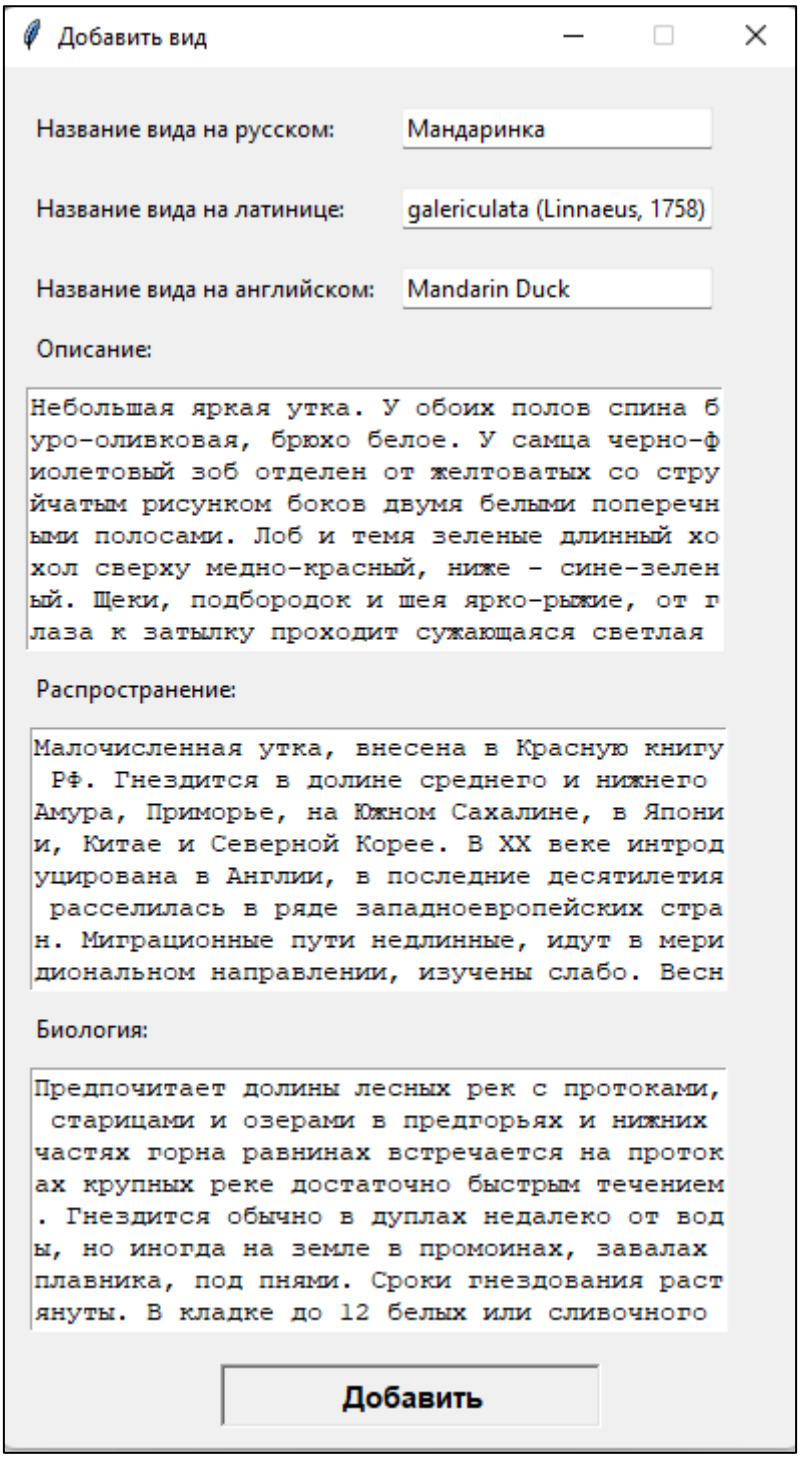

Рисунок 5: окно «Добавить вид».

Кнопка «Добавить категорию» открывает окно, в котором вносятся данные для категории и добавляются в таблицу с категорией и отображается в таблице на главной странице.

В окне «Добавить категорию» расположены:

- кнопка «Выбрать файл» выбирается картинка для пред показа вида в категории;
- Выпадающий список, хранящий в себе добавленные виды;
- Поле даты для.
- Кнопка «Добавить» собирает данные и добавляет в таблицу для категорий, которые отображаются в таблице на главном окне.

```
name_values = self.get_name()
        self.entry_species = ttk.Combobox(self,
values=name_values)
        self.entry_species.place(x=200, y=50)

btn_file = ttk.Button(self, text='Выбрать файл',
command=self.open_file_path)
        btn_file.place(x=240, y=20)

btn_ok.bind('<Button-1>', lambda event:
self.veiw.records(self.entry_species.get(),

self.entry_date.get(),

self.get_timestamp_from_string(self.entry_date.get()),

self.entry_path.cget("text")))


def open_file_path(self):
        filetypes = (('Image', '*.jpg'), ('All files',
'*.*'))
        filename = fd.askopenfilename(title='Open a
file', initialdir='/', filetypes=filetypes)
        self.entry_path = ttk.Label(self,
text=filename)
        self.entry_path.place(x=100, y=20)
```

Листинг 5

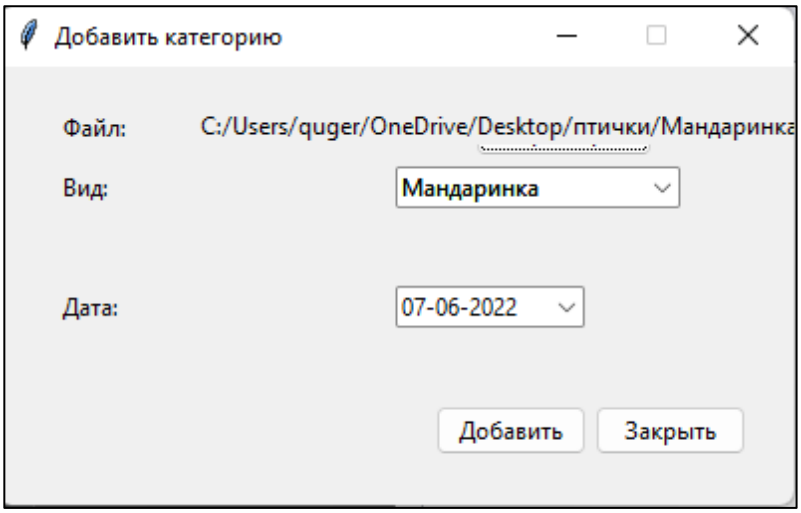

Рисунок 6: окно «Добавить категорию».

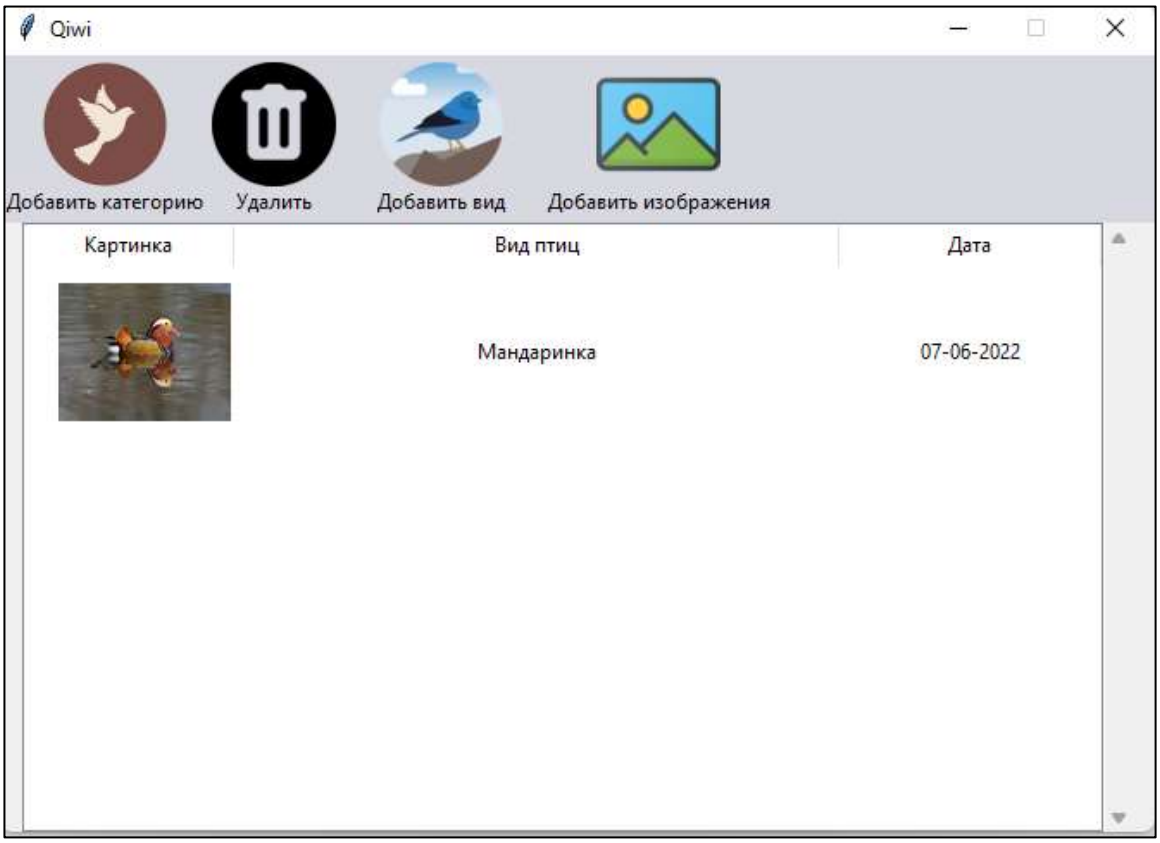

Рисунок 7: главное окно после добавления категории.

Кнопка «Добавить изображение» открывает окно, в котором добавляются фотографии к виду.

В окне «Добавить категорию» расположены:

- Кнопка для выбора файла с изображением;
- Выпадающий список с выбором виду, к которому добавить изображение;
- Кнопка добавить, которая собирает выбранные данные и добавляет в таблицу с изображениями.

```
name_values = self.get_name()
        self.entry_species = ttk.Combobox(self,
values=name_values,width=30)
        self.entry_species.place(x=150, y=50)

        btn_file = ttk.Button(self, text='Выбрать
файл', command=self.open_file_path)
        btn_file.place(x=240, y=20)

btn_ok = ttk.Button(self, text='Добавить')
        btn_ok.place(x=220, y=170)
        btn_ok.bind('<Button-1>', lambda event:
self.veiw.records_for_picture(self.entry_species.get()
,
self.entry_path.cget("text")))
```

Листинг 6.

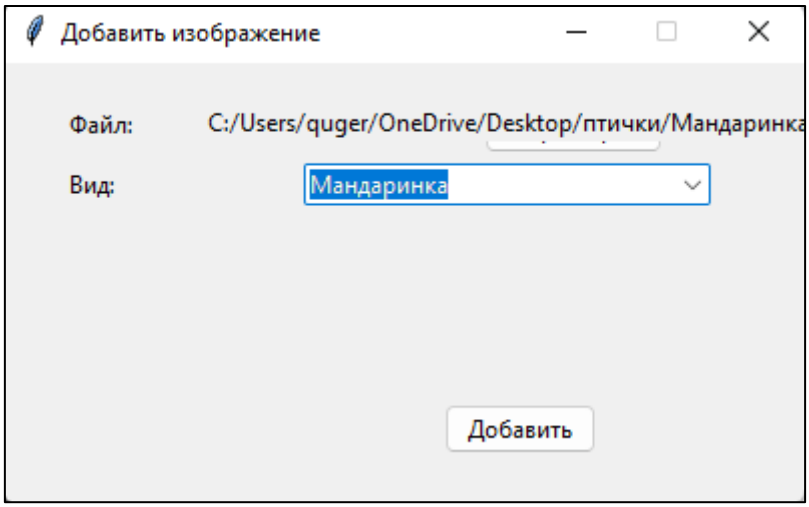

Рисунок 8: окно «Добавить изображение».

По двойному щелчку по выбранному элементу из таблицы на главном экране откроется окно, в котором собрана вся информация по виду и фотографии.

```
def OnDoubleClick(self, event):
        item = self.tree.identify('item', event.x,
event.y)
        self.open_dialog_view_page()


def init_veiw_page(self):
        self.title('Галерея')
        self.geometry('1000x1000+400+1')
        self.resizable(False, False)

        data = self.view.get_data_species()

        self.db.c.execute('''SELECT id, pic_min,
pic_mid FROM picture WHERE name=?''', (data[0],))
        e = self.get_len_data(data[3])
        f = self.db.c.fetchall()
        self.db.conn.commit()

        self.a = self.get_list_pic_mid(f)

        self.imglist_mid_pic = []
        for record in f:
            img = ImageTk.PhotoImage(data=record[1])
            self.imglist_mid_pic.append(img)

        lbl_name_ru = ttk.Label(self, text=data[0],
                                font=("Arial", 28))
        lbl_name_ru.place(x=40, y=10)

        lbl_name_lat = ttk.Label(self, text=data[1],
                                font=("Arial", 16))
        lbl_name_lat.place(x=40, y=60)

        lbl_name_en = ttk.Label(self, text=data[2],
                                 font=("Arial", 14))
        lbl_name_en.place(x=40, y=90)

        self.Artwork = tk.Label(self, image=self.a[0],
background="black").place(x=40, y=120)
```

```
        self.creat_btn(f)

        lbl_notes = ttk.Label(self, text="Описание:",
font=("Arial", 14))
        lbl_notes.place(x=40, y=610)

        lbl_notes_text = ttk.Label(self, text=data[3],
width= 130, font=("Arial", 10))
        lbl_notes_text.place(x=40, y=640)
        lbl_notes_text.bind('<Configure>',  lambda e:
lbl_notes_text.config(wraplength=lbl_notes_text.winfo_
width()))

        lbl_place = ttk.Label(self,
text="Расположение:", font=("Arial", 14))
        lbl_place.place(x=40, y=730 + e)

        lbl_place_text = ttk.Label(self, text=data[4],
width=130, font=("Arial", 10))
        lbl_place_text.place(x=40, y=760 + e)
        lbl_place_text.bind('<Configure>', lambda e:
lbl_place_text.config(wraplength=lbl_place_text.winfo_
width()))

        lbl_biologi = ttk.Label(self,
text="Биология:",font=("Arial", 14))
        lbl_biologi.place(x=40, y=810 + e)

        lbl_biologi_text = ttk.Label(self,
text=data[5], width=130, font=("Arial", 10))
        lbl_biologi_text.place(x=40, y=840 + e)
        lbl_biologi_text.bind('<Configure>', lambda e:
lbl_biologi_text.config(wraplength=lbl_biologi_text.wi
nfo_width()))
```

Листинг 7

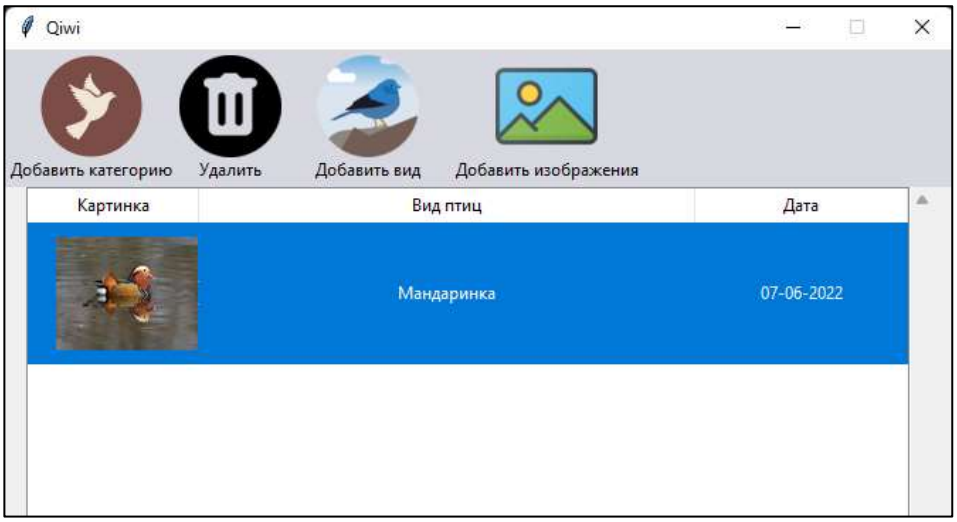

Рисунок 9: выбранный элемент

Рисунок 10: окно «Галерея»

Если нажать на маленькое изображение оно отобразится в большем размере.

Рисунок 11

Рисунок 12

Заключение

В рамках курсовой работы мною был проведен анализ предметной области, поставленной задачи и имеющихся решений. Разработана архитектура системы и модель интерфейса. Реализована персональная база данных, как часть будущей системы каталогизатора.

В ходе работы были повышены навыки в программирование на Python и в использовании библиотеки tkinter и SQL запросов.

В дальнейшем разработка будет производиться с помощью другого графического интерфейса так как библиотека tkinter имеет ограниченный ресурс в сравнение с другими библиотеками.

Список литературы

1. Советов, Б. Я. Базы данных : учебник для среднего профессионального образования / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. — 3-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2020. — 420 с. — (Профессиональное образование). — ISBN 978-5-534-09324-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/453635 (дата обращения: 08.06.2022).
2. Сузи Р.А. Язык программирования Python : учебное пособие / Сузи Р.А.. — Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 350 c. — ISBN 978-5-4497-0705-5. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/97589.html (дата обращения: 08.06.2022). — Режим доступа: для авторизир. Пользователей
3. Маккинли Уэс Python и анализ данных / Маккинли Уэс. — Саратов : Профобразование, 2019. — 482 c. — ISBN 978-5-4488-0046-7. — Текст : электронный // IPR SMART : [сайт]. — URL: https://www.iprbookshop.ru/88752.html (дата обращения: 09.06.2022). — Режим доступа: для авторизир. пользователей

Список интернет-источников

https://docs.python.org/3/library/sqlite3.html

https://docs.python.org/3/library/tkinter.html

Приложение

```
import sqlite3
import datetime
import tkinter as tk
import os
from tkinter import ttk
from tkcalendar import DateEntry, Calendar
from tkinter import filedialog as fd
from io import BytesIO
from PIL import ImageTk, Image

class Main(tk.Frame):
    def __init__(self, root):
        super().__init__(root)
        self.init_main()
        self.db = db
        self.view_records()

    def init_main(self):
        # СОЗДАНИЕ ПАНЕЛИ КНОПОК
        toolbar = tk.Frame(bg='#d7d8e0', bd=2)
        toolbar.pack(side=tk.TOP, fill=tk.X)

        style = ttk.Style()
        style.configure("Treeview", rowheight=100)

        # СОЗДАНИЕ КНОПКИ С КАРТИНКОЙ ДОБАВИТЬ ИЗОБРАЖЕНИЕ
        self.add_img = tk.PhotoImage(file="icon/add_icon.png")
        btn_open_dialog = tk.Button(toolbar, text='Добавить
категорию', command=self.open_dialog, bg='#d7d8e0', bd=0,
                                    compound=tk.TOP,
image=self.add_img)
        btn_open_dialog.pack(side=tk.LEFT)

        # СОЗДАНИЕ КНОПКИ С КАРТИНКОЙ УДАЛИТЬ
        self.delete_img = tk.PhotoImage(file='icon/trash.png')
        btn_delete = tk.Button(toolbar, text='Удалить',
bg='#d7d8e0', bd=0, image=self.delete_img,
                               compound=tk.TOP,
command=self.delete_records)
        btn_delete.pack(side=tk.LEFT)

        # СОЗДАНИЕ КНОПКИ ДОБАВИТЬ ВИД
        self.species = tk.PhotoImage(file='icon/sp.png')
        btn_add_species = tk.Button(toolbar, text='Добавить
вид', bg='#d7d8e0', bd=0, image=self.species,
                               compound=tk.TOP,
command=self.open_dialog_species)
        btn_add_species.pack(side=tk.LEFT, padx=20)

        self.images = tk.PhotoImage(file='icon/image.png')
```

```
        images = tk.Button(toolbar, text='Добавить
изображения', bg='#d7d8e0', bd=0, image=self.images,
                                        compound=tk.TOP,
command=self.open_dialog_add_images_page)
        images.pack(side=tk.LEFT)

        # СОЗДАНИЕ ТАБЛИЦЫ С КОЛОНКАМИ
        self.tree = ttk.Treeview(self, columns=('species',
'data'))
        self.tree.column("species", width=350,
anchor=tk.CENTER)
        self.tree.column("data", width=150, anchor=tk.CENTER)
        self.tree.column("#0", width=120)

        # ЗАГОЛОВКИ
        self.tree.heading("species", text='Вид птиц')
        self.tree.heading("data", text='Дата')
        self.tree.heading("#0", text="Картинка")

        # УПАКОВКА ТАБЛИЦЫ
        self.tree.pack(side=tk.LEFT)
        self.tree.bind("<Double-1>", self.OnDoubleClick)

        scroll = tk.Scrollbar(self, command=self.tree.yview)
        scroll.pack(side=tk.RIGHT, fill=tk.Y)
        self.tree.configure(yscrollcommand=scroll.set)

    # УДАЛЕНИЕ ИЗ ТАБЛИЦЫ И БД
    def delete_records(self):
        for selection_item in self.tree.selection():
            self.db.c.execute('''DELETE FROM birds WHERE
species=?''', (self.tree.set(selection_item, '#1'),))
        self.db.conn.commit()
        self.view_records()

    def OnDoubleClick(self, event):
        item = self.tree.identify('item', event.x, event.y)
        self.open_dialog_view_page()

    def get_data_species(self):
        self.db.c.execute('''SELECT name, name_lat, name_ing,
notes, places , biologi FROM species WHERE name=?''',

(self.tree.set(self.tree.selection()[0], '#1'),))
        data_kek = []
        for row in self.db.c.fetchall():
            data_kek.append(row)
        lst_data = list(data_kek[0])
        return lst_data

    def records_for_speces(self, name, name_lat, name_ing,
notes, places , biologi):
```

```
        self.db.insert_speceis(name, name_lat, name_ing, notes,
places , biologi)

    # МЕТОД ДЛЯ ЗАПИСИ В БД И ОТОПБРАЖЕНИЯ В ТАБЛИЦЕ
    def records(self, species, date_str, date_int, pic):
        self.db.insert_data(species, date_str, date_int, pic)
        self.view_records()

    def records_for_picture(self, name, pic):
        self.db.insert_picture(name, pic)

    # МЕТОД ОТОБРАЖЕНИЯ ДАННЫХ
    def view_records(self):
        self.db.c.execute('''SELECT pic, species, date_str FROM
birds''')
        [self.tree.delete(i) for i in self.tree.get_children()]
        self.imglist = []
        for record in self.db.c.fetchall():
            img = ImageTk.PhotoImage(data=record[0])

            self.tree.insert("", 'end', image=img,
values=record[1:])
            self.imglist.append(img)

    def open_dialog(self):
        Child()

    def open_dialog_species(self):
        Speceis()

    def open_dialog_view_page(self):
        VeiwPage()

    def open_dialog_add_images_page(self):
        Picture()

class Picture(tk.Toplevel):
    def __init__(self):
        super().__init__(root)
        self.veiw = app
        self.db = db
        self.init_child()

    def init_child(self):
        # СОЗДАНИЕ ДИАЛОГОВОГО ОКНА
        self.title('Добавить изображение')
        self.geometry('400x220+400+300')
        self.resizable(False, False)

        # СОЗДАНИЕ ЛЕЙБЛОВ ДЛЯ ПОДПИСИ
        label_path = tk.Label(self, text='Файл:')
        label_path.place(x=30, y=20)
```

```
        label_species = tk.Label(self, text='Вид:')
        label_species.place(x=30, y=50)

        name_values = self.get_name()
        self.entry_species = ttk.Combobox(self,
values=name_values,width=30)
        self.entry_species.place(x=150, y=50)

        btn_file = ttk.Button(self, text='Выбрать файл',
command=self.open_file_path)
        btn_file.place(x=240, y=20)

        btn_ok = ttk.Button(self, text='Добавить')
        btn_ok.place(x=220, y=170)
        btn_ok.bind('<Button-1>', lambda event:
self.veiw.records_for_picture(self.entry_species.get(),

self.entry_path.cget("text")))

        # ПРИОРИТЕТ ДЛЯ 2-ГО ОКНА
        self.grab_set()
        self.focus_set()


    def get_name(self):
        self.db.c.execute('''SELECT name FROM species''')
        data_name = []
        for row in self.db.c.fetchall():
            data_name.append(row[0])
        return data_name

    def open_file_path(self):
        filetypes = (('Image', '*.jpg'), ('All files', '*.*'))
        filename = fd.askopenfilename(title='Open a file',
initialdir='/', filetypes=filetypes)
        os.path.basename(filename)
        self.entry_path = ttk.Label(self, text=filename)
        self.entry_path.place(x=100, y=20)

class Child(tk.Toplevel):
    def __init__(self):
        super().__init__(root)
        self.veiw = app
        self.db = db
        self.init_child()

    def init_child(self):
        # СОЗДАНИЕ ДИАЛОГОВОГО ОКНА
        self.title('Добавить изображение')
        self.geometry('400x220+400+300')
        self.resizable(False, False)
```

```
        # СОЗДАНИЕ ЛЕЙБЛОВ ДЛЯ ПОДПИСИ
        label_path = tk.Label(self, text='Файл:')
        label_path.place(x=30, y=20)

        label_species = tk.Label(self, text='Вид:')
        label_species.place(x=30, y=50)

        # label_place = tk.Label(self, text='Место съемки:')
        # label_place.place(x=30, y=80)

        label_select = tk.Label(self, text='Дата:')
        label_select.place(x=30, y=110)

        # ПОЛЯ ДЛЯ ВВОДА ДАННЫХ
        self.entry_path = ttk.Label(self, text='тут путь к
файлу')
        self.entry_path.place(x=100, y=20)

        name_values = self.get_name()
        self.entry_species = ttk.Combobox(self,
values=name_values)
        self.entry_species.place(x=200, y=50)

        # self.entry_place = ttk.Entry(self)
        # self.entry_place.place(x=200, y=80)

        self.entry_date = DateEntry(self, date_pattern='dd-mm-
YYYY')
        self.entry_date.place(x=200, y=110)

        #КНОПОКИ ДЛЯ ДИАЛОГОВОГО ОКНА
        btn_file = ttk.Button(self, text='Выбрать файл',
command=self.open_file_path)
        btn_file.place(x=240, y=20)

        btn_cancel = ttk.Button(self, text='Закрыть',
command=self.destroy)
        btn_cancel.place(x=300, y=170)

        btn_ok = ttk.Button(self, text='Добавить')
        btn_ok.place(x=220, y=170)
        btn_ok.bind('<Button-1>', lambda event:
self.veiw.records(self.entry_species.get(),

self.entry_date.get(),

self.get_timestamp_from_string(self.entry_date.get()),

self.entry_path.cget("text")))

        # ПРИОРИТЕТ ДЛЯ 2-ГО ОКНА
```

```
        self.grab_set()
        self.focus_set()

    def get_name(self):
        self.db.c.execute('''SELECT name FROM species''')
        data_name = []
        for row in self.db.c.fetchall():
            data_name.append(row[0])
        return data_name

    # ПОЛУЧЕНИЕ ДАТЫ В ВИДЕ INTEGER
    def get_timestamp(self, y, m, d):
        return
int(datetime.datetime.timestamp(datetime.datetime(y, m, d)))

    # ПОЛУЧЕНИЕ ДАТЫ ИЗ СТРОКИ В INTEGER
    def get_timestamp_from_string(self, s):
        t = s.split('-')
        return self.get_timestamp(int(t[2]), int(t[1]),
int(t[0]))

    # ОТКРЫВАЕТ ОКНО
    def open_file_path(self):
        filetypes = (('Image', '*.jpg'), ('All files', '*.*'))
        filename = fd.askopenfilename(title='Open a file',
initialdir='/', filetypes=filetypes)
        self.entry_path = ttk.Label(self, text=filename)
        self.entry_path.place(x=100, y=20)

class VeiwPage(tk.Toplevel):
    def __init__(self):
        super().__init__()
        self.view = app
        self.db = db
        self.init_veiw_page()

    def init_veiw_page(self):
        self.title('Галерея')
        self.geometry('1000x1000+400+1')
        self.resizable(False, False)

        data = self.view.get_data_species()
        # print(data[0])
        self.db.c.execute('''SELECT id, pic_min, pic_mid FROM
picture WHERE name=?''', (data[0],))
        e = self.get_len_data(data[3])
        f = self.db.c.fetchall()
        self.db.conn.commit()

        self.a = self.get_list_pic_mid(f)

        self.imglist_mid_pic = []
```

```
        for record in f:
            img = ImageTk.PhotoImage(data=record[1])
            self.imglist_mid_pic.append(img)

        lbl_name_ru = ttk.Label(self, text=data[0],
                                font=("Arial", 28))
        lbl_name_ru.place(x=40, y=10)

        lbl_name_lat = ttk.Label(self, text=data[1],
                                font=("Arial", 16))
        lbl_name_lat.place(x=40, y=60)

        lbl_name_en = ttk.Label(self, text=data[2],
                                 font=("Arial", 14))
        lbl_name_en.place(x=40, y=90)

        self.Artwork = tk.Label(self, image=self.a[0],
background="black").place(x=40, y=120)

        self.creat_btn(f)

        lbl_notes = ttk.Label(self, text="Описание:",
font=("Arial", 14))
        lbl_notes.place(x=40, y=610)

        lbl_notes_text = ttk.Label(self, text=data[3], width=
130, font=("Arial", 10))
        lbl_notes_text.place(x=40, y=640)
        lbl_notes_text.bind('<Configure>',  lambda e:
lbl_notes_text.config(wraplength=lbl_notes_text.winfo_width()))

        lbl_place = ttk.Label(self, text="Расположение:",
font=("Arial", 14))
        lbl_place.place(x=40, y=730 + e)

        lbl_place_text = ttk.Label(self, text=data[4],
width=130, font=("Arial", 10))
        lbl_place_text.place(x=40, y=760 + e)
        lbl_place_text.bind('<Configure>', lambda e:
lbl_place_text.config(wraplength=lbl_place_text.winfo_width()))

        lbl_biologi = ttk.Label(self,
text="Биология:",font=("Arial", 14))
        lbl_biologi.place(x=40, y=810 + e)

        lbl_biologi_text = ttk.Label(self, text=data[5],
width=130, font=("Arial", 10))
        lbl_biologi_text.place(x=40, y=840 + e)
        lbl_biologi_text.bind('<Configure>', lambda e:
lbl_biologi_text.config(wraplength=lbl_biologi_text.winfo_width
()))
```

```
        if len(data[4])  == 0:
            lbl_place.destroy()

        if len(data[5]) == 0:
            lbl_biologi.destroy()

        print(len(data[4]))

    def creat_btn(self, fetch):
        self.list_pic = []
        r = 0
        for record in fetch:
            img = ImageTk.PhotoImage(data=record[1])
            self.list_pic.append(img)

        list_btn = []

        bt = tk.Button(self, compound=tk.TOP, font=('Arial',
10), bd=0)
        bt.bind("<Button-1>", lambda event: tk.Label(self,
image=self.a[0], background="black").place(x=40, y=120))
        bt.place(x=700, y=120)
        list_btn.append(bt)

        bt1 = tk.Button(self, compound=tk.TOP, font=('Arial',
10),bd=0)
        bt1.bind("<Button-1>", lambda event: tk.Label(self,
image=self.a[1], background="black").place(x=40, y=120))
        bt1.place(x=700, y=220)
        list_btn.append(bt1)

        bt2 = tk.Button(self, compound=tk.TOP, font=('Arial',
10),bd=0)
        bt2.bind("<Button-1>", lambda event: tk.Label(self,
image=self.a[2], background="black").place(x=40, y=120))
        bt2.place(x=700, y=320)
        list_btn.append(bt2)

        bt3 = tk.Button(self, compound=tk.TOP, font=('Arial',
10),bd=0)
        bt3.bind("<Button-1>", lambda event: tk.Label(self,
image=self.a[3], background="black").place(x=40, y=120))
        bt3.place(x=700, y=420)
        list_btn.append(bt3)

        for i in range(len(self.list_pic)):
            list_btn[i] =
list_btn[i].configure(image=self.list_pic[i])

    def get_list_pic_min(self, fetch):
        self.list_pic = []
        r = 0
```

```
        for record in fetch:
            img = ImageTk.PhotoImage(data=record[1])
            self.list_pic.append(img)
        b = []

        for i in range(len(self.list_pic)):
            bt = tk.Button(self,compound=tk.TOP,
image=self.list_pic[i], font=('Arial', 10))
            bt.bind("<Button-1>", lambda event: tk.Label(self,
image=self.a[i], background="black").place(x=40, y=120))
            bt.place(x=700, y=120 + r)
            b.append(bt)
            r += 120
        for i in range(len(b)):
            b[i] = b[i].configure(text=str(i))

    def get_list_pic_mid(self, fetch):
        self.list_pic = []
        r = 0
        for record in fetch:
            img = ImageTk.PhotoImage(data=record[2])
            self.list_pic.append(img)
        return self.list_pic

    def get_len_data(self, data):
        e = 0
        if len(data) > 450:
            e = 40
        return e

class Speceis(tk.Toplevel):
    def __init__(self):
        super().__init__()
        self.init_speceis()
        self.view = app

    def init_speceis(self):
        self.title('Добавить вид')
        self.geometry('400x690+800+200')
        self.resizable(False, False)

        label_speceis = tk.Label(self, text='Название вида на
русском:')
        label_speceis.place(x=15, y=20)
        self.entry_species = ttk.Entry(self, width=25)
        self.entry_species.place(x=200, y=20)


        label_speceis_lat = tk.Label(self, text='Название вида
на латинице:')
        label_speceis_lat.place(x=15, y=60)
        self.entry_species_lat = ttk.Entry(self, width=25)
```

```
        self.entry_species_lat.place(x=200, y=60)


        label_speceis_en = tk.Label(self, text='Название вида
на английском:')
        label_speceis_en.place(x=15, y=100)
        self.entry_species_en = ttk.Entry(self, width=25)
        self.entry_species_en.place(x=200, y=100)

        label_notes = tk.Label(self, text='Описание:')
        label_notes.place(x=15, y=130)
        self.text_notes = tk.Text(self,width=43, height=8)
        self.text_notes.place(x=13, y=160)

        label_places = tk.Label(self, text='Распространение:')
        label_places.place(x=15, y=300)
        self.text_places = tk.Text(self, width=43, height=8)
        self.text_places.place(x=15, y=330)

        label_biologi = tk.Label(self, text='Биология:')
        label_biologi.place(x=15, y=470)
        self.text_biologi= tk.Text(self, width=43, height=8)
        self.text_biologi.place(x=15, y=500)

        btn_submit = tk.Button(self, text='Добавить',
width=20, font=('Helvetica', 11, 'bold'))
        btn_submit.place(x=110, y=648)
        btn_submit.bind('<Button-1>', lambda event:
self.view.records_for_speces(self.entry_species.get(),

self.entry_species_lat.get(),

self.entry_species_en.get(),

self.text_notes.get(1.0, 'end-1c'),

self.text_places.get(1.0, 'end-1c'),

self.text_biologi.get(1.0, 'end-1c')))

        self.grab_set()
        self.focus_set()

class DB:
    # СОЗДАНИЕ БД
    def __init__(self):
        self.conn = sqlite3.connect('db/database.db')
        self.c = self.conn.cursor()
        self.c.execute(
            '''CREATE TABLE IF NOT EXISTS birds (
            id integer primary key, species text, date_str,
date_int integer, place text, pic blob, notes text)''')
```

```
        self.conn.commit()

        # СОЗДАНИЕ БД
        self.conn = sqlite3.connect('db/database.db')
        self.c = self.conn.cursor()
        self.c.execute(
            '''CREATE TABLE IF NOT EXISTS species (
            species_id integer primary key, name text, name_lat
text, name_ing text, notes text, places text, biologi text, pic
blob)''')
        self.conn.commit()

        self.conn = sqlite3.connect('db/database.db')
        self.c = self.conn.cursor()
        self.c.execute(
            '''CREATE TABLE IF NOT EXISTS picture (
            id integer primary key, name text, pic blob,
pic_min blob, pic_mid blob)''')
        self.conn.commit()

    # ВСТАВКА ДАННЫХ ИЗ ФОРМЫ
    def insert_data(self, species, date_str, date_int, pic):
        img = Image.open(pic)
        img = img.resize((100, 80), Image.ANTIALIAS)
        with BytesIO() as f:
            img.save(f, 'PNG')
            self.fob = f.getvalue()
        self.c.execute('''INSERT INTO birds(species, date_str,
date_int, pic) VALUES (?, ?, ?, ?)''',
                       (species, date_str, date_int, self.fob))
        self.conn.commit()

    def insert_speceis(self, name, name_lat, name_ing, notes,
places , biologi):
        self.c.execute('''INSERT INTO species (name, name_lat,
name_ing, notes, places , biologi) VALUES (?, ?, ?, ?, ?,
?)''',
                       (name, name_lat, name_ing, notes, places
, biologi))
        self.conn.commit()

    def insert_picture(self,name, pic):
        img_norm = Image.open(pic)

        img_min = Image.open(pic)
        img_min = img_min.resize((120, 90))

        img_mid= Image.open(pic)
        img_mid = img_mid.resize((640, 480))

        with BytesIO() as f:
            img_norm.save(f, 'PNG')
```

```
            self.fob = f.getvalue()

        with BytesIO() as f:
            img_min.save(f, 'PNG')
            self.fob_min = f.getvalue()

        with BytesIO() as f:
            img_mid.save(f, 'PNG')
            self.fob_mid = f.getvalue()

        print(self.fob)
        self.c.execute('''INSERT INTO picture(name,
pic,pic_min, pic_mid) VALUES (?, ?, ?, ?)''',
                       (name, self.fob, self.fob_min,
self.fob_mid))
        self.conn.commit()
        print("add")

if __name__ == "__main__":
    root = tk.Tk()
    db = DB()
    app = Main(root)
    app.pack()
    root.title("Qiwi ")
    root.geometry("670x450+600+200")
    root.resizable(False, False)
    root.mainloop()
```