

Санкт-Петербургский Национальный  
Исследовательский Университет  
Информационных технологий, механики и оптики

**Лабораторная работа 2**  
**Программа для учета грузового**  
**транспорта**

Выполнил: Буров  
Глеб  
Группа № К3123  
Проверила: Казанова  
Полина Петровна

## СОДЕРЖАНИЕ

	Стр.
ЦЕЛЬ РАБОТЫ.....	3
ЗАДАЧИ .....	4
1   Ход работы.....	5
1.1   Описание этапа анализа предметной области и требований .....	5
1.2   Реализация базы данных.....	5
1.3   Описание классов, методов и реализации интерфейса.....	6
ЗАКЛЮЧЕНИЕ.....	19

## ЦЕЛЬ РАБОТЫ

Целью работы является создание программного обеспечения учета грузового транспорта для Автотранспортного отдела логистической компании. Приложение должно подбирать доступный грузовой транспорт в зависимости от размера (веса) груза, реализовать ПО с помощью ООП, БД, графического интерфейса.

## ЗАДАЧИ

В приложении необходимо реализовать следующие функции:

1. Добавление/удаление грузового транспорта
2. Просмотр всего доступного транспорта
3. Просмотр грузового транспорта по грузоподъемности
4. Просмотр свободного грузового транспорта
5. Внесение заявки на перевоз груза по указанным габаритам
6. Подбор и бронирование свободного транспорта
7. Просмотр занятого грузового транспорта
8. Сохранение данных в базу данных

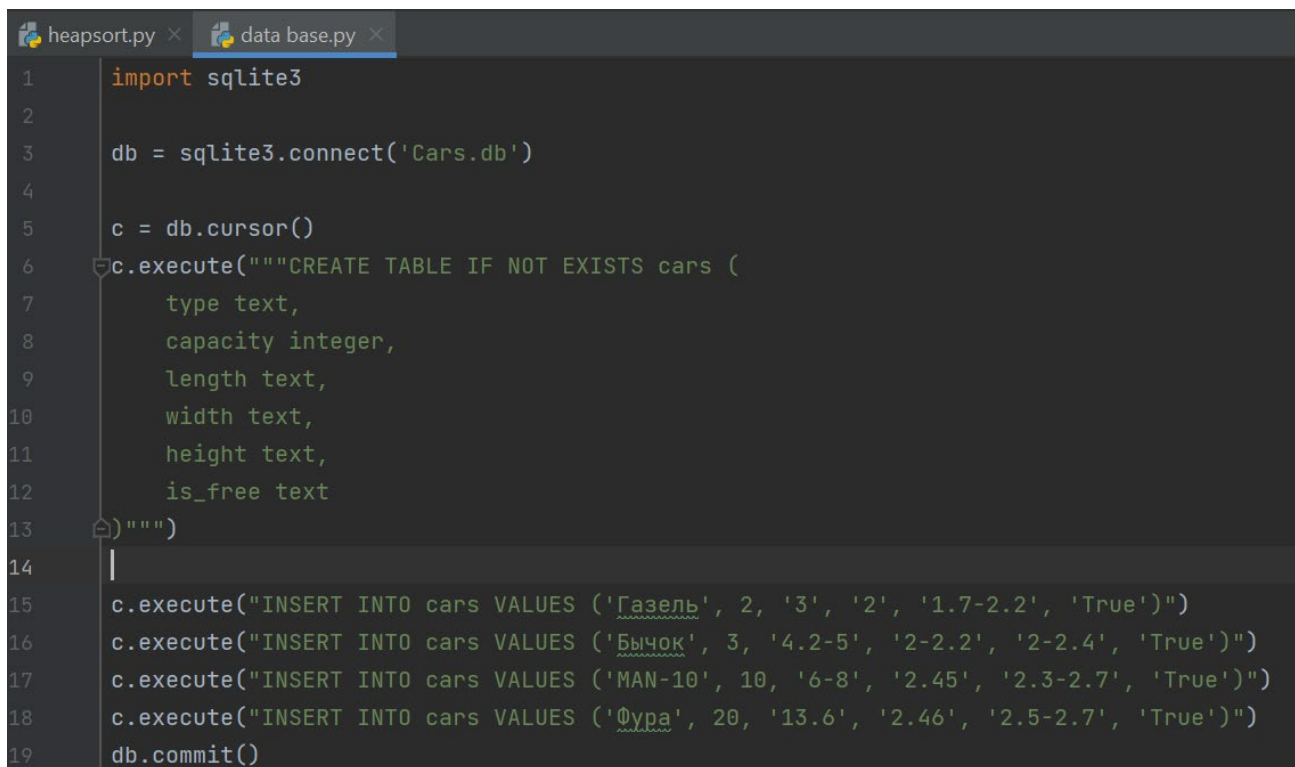
## 1 Ход работы

### 1.1 Описание этапа анализа предметной области и требований

Приложение предназначено для контроля собственных денежных средств. Для работы требовалось написать программу на языке python, которая будет удовлетворять всем требованиям. Основные функции – создание и удаление заказа, добавление и удаление транспорта, возможность просмотра транспорта в нескольких режимах (“Весь транспорт”, “Свободный транспорт”, “Занятый транспорт”, “По грузоподъемности”). Необходимо реализовать сохранение данных в базу данных. Для приложения должен быть реализован графический интерфейс.

### 1.2 Реализация базы данных

Для реализации базы данных использовалась встроенная библиотека sqlite3. В отдельном файле “data base.py” была создана база данных Cars.db с таблицей cars, содержащей в себе следующие параметры: type, capacity, length, width, height, is\_free. После этого в таблицу были занесены начальные значения (см. Рисунок 1).



```
1 import sqlite3
2
3 db = sqlite3.connect('Cars.db')
4
5 c = db.cursor()
6 c.execute("""CREATE TABLE IF NOT EXISTS cars (
7     type text,
8     capacity integer,
9     length text,
10    width text,
11    height text,
12    is_free text
13)""")
14
15 c.execute("INSERT INTO cars VALUES ('Газель', 2, '3', '2', '1.7-2.2', 'True')")
16 c.execute("INSERT INTO cars VALUES ('Бычок', 3, '4.2-5', '2-2.2', '2-2.4', 'True')")
17 c.execute("INSERT INTO cars VALUES ('MAN-10', 10, '6-8', '2.45', '2.3-2.7', 'True')")
18 c.execute("INSERT INTO cars VALUES ('Фурга', 20, '13.6', '2.46', '2.5-2.7', 'True')")
19 db.commit()
```

Рисунок 1– Создание таблицы cars

### 1.3 Описание классов, методов и реализации интерфейса

Два класса, использующихся в программе – Car и Application. Класс Car необходим для создания экземпляров транспортных средств, имеющих в автопарке (см. Рисунок 2). Имеется конструктор `__init__`, которому передаются параметры `type_car` (в нашем случае это название автомобиля), `capacity` (грузоподъемность), `length` (допустимая длина груза, может быть задана интервалом), `width` (допустимая ширина груза, может быть задана интервалом), `height` (допустимая высота груза, также может быть задана интервалом), `is_free` (свободна ли машина в данный момент).

```
class Car:
    def __init__(self, type_car, capacity, length, width, height, is_free):
        self.type_car = type_car
        self.capacity = capacity
        self.length = length
        self.width = width
        self.height = height
        self.is_free = is_free
```

Рисунок 2 – Класс Car

Класс Application отвечает за наше приложение, реализацию функций и графический интерфейс, которое реализуется с помощью библиотеки tkinter. Класс имеет свои атрибуты: список `transport`, в котором будут храниться экземпляры класса Car, имеющиеся в наличии транспортные средства, `cur_labels`

и список labels – счетчик лейблов, которые расположены в окне приложения (необходимо для просмотра по дате/грузоподъемности и т.д.) (см. Рисунок 3).

```
class Application:
    window = tk.Tk()
    window.title('Программное обеспечение')
    transport = []
    cur_labels = 0
    view = tk.IntVar()
    labels = []
```

Рисунок 3 – Атрибуты класса Application

Рассмотрим подробнее методы, реализованные в классе Application. Метод start (см. Рисунок 4) запускает отображение окна приложения, а также вызывает метод initialization.

```
def start(self):
    self.initialization()
    Application.window.mainloop()
```

Рисунок 4 – Метод start

В методе initialization (см. Рисунок 5) происходит запрос в базу данных Cars, после чего создаются экземпляры класса Car с информацией из базы. Созданные экземпляры добавляются в Application.transport. После чего вызывается метод create\_widgets.

```
def initialization(self):
    db = sqlite3.connect('Cars.db')
    cursor = db.cursor()
    cars = list(cursor.execute("SELECT * FROM cars"))
    for car in cars:
        Application.transport.append(Car(car[0], car[1], car[2], car[3], car[4], car[5]))
    db.close()
    self.create_widgets()
```

Рисунок 5 – метод initialization

Метод create\_widgets (см. Рисунок 6) создает виджеты, которые являются заголовками столбцов таблицы, отображающейся в окне приложения (см.

Рисунок 7). Также в окне располагаются кнопки “Создать заказ”, “Удалить заказ”, “Добавить транспорт”, “Удалить транспорт”. Сопровождается вызовом метода `place_cars`, в который передается список `Application.transport` с имеющимися в наличии ТС.

```
def create_widgets(self):
    number_of_car = tk.Label(text='№', state='disabled', justify='center', disabledforeground='black', width=10,
                             borderwidth=1, relief='solid', font='Arial 10 bold')
    number_of_car.grid(row=6, column=0)
    type_of_car = tk.Label(text='Тип', state='disabled', justify='center', disabledforeground='black', width=20,
                           borderwidth=1, relief='solid', font='Arial 10 bold')
    type_of_car.grid(row=6, column=1)

    capacity = tk.Label(text='Грузоподъемность, тонн', state='disabled', justify='center',
                        disabledforeground='black', borderwidth=1, relief='solid', width=25, font='Arial 10 bold')
    capacity.grid(row=6, column=2)

    length = tk.Label(text='Длина', state='disabled', justify='center', disabledforeground='black', width=20,
                      borderwidth=1, relief='solid', font='Arial 10 bold')
    length.grid(row=6, column=3)

    width = tk.Label(text='Ширина', state='disabled', justify='center', disabledforeground='black', width=20,
                     borderwidth=1, relief='solid', font='Arial 10 bold')
    width.grid(row=6, column=4)

    height = tk.Label(text='Высота', state='disabled', justify='center', disabledforeground='black', width=20,
                      borderwidth=1, relief='solid', font='Arial 10 bold')
    height.grid(row=6, column=5)

    status = tk.Label(text='Статус', state='disabled', justify='center', disabledforeground='black', width=20,
                      borderwidth=1, relief='solid', font='Arial 10 bold')
    status.grid(row=6, column=6)

    tk.Button(text='Создать заказ', command=self.create_order).grid(row=2, column=1, padx=5, pady=5)
    tk.Button(text='Удалить заказ', command=self.remove_order).grid(row=2, column=2, padx=5, pady=5)
    tk.Button(text='Добавить транспорт', command=self.add_car).grid(row=3, column=1, padx=5, pady=5)
    tk.Button(text='Удалить транспорт', command=self.remove_car).grid(row=3, column=2, padx=5, pady=5)
    self.view.set(1)
    cur_row = 1
    for mode in sorted(view_mode):
        tk.Radiobutton(text=view_mode[mode], variable=self.view, value=mode, font='Arial 9',
                       command=self.sorting).grid(row=cur_row,
                                                  column=4)
        cur_row += 1
    self.place_cars(Application.transport)
```

Рисунок 6 – Метод `create_widgets`

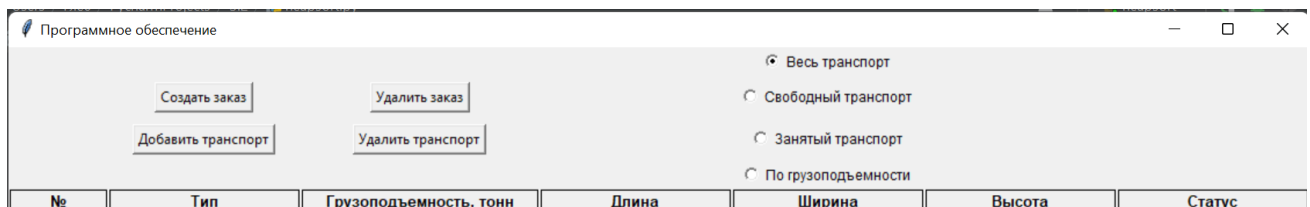


Рисунок 7 – Виджеты

Следующий метод – `place_cars` – является статическим (см. Рисунок 8). Он получает список экземпляров класса `Car` и размещает информацию о транспортных средствах в виде лейблов. Для этого используется вспомогательная переменная – `cur_row`, обозначающая текущий ряд, на котором будут размещаться лейблы. После создания и размещения с помощью метода `grid` экземпляры класса `Label` добавляются в список `Application.labels`.



```

307 @staticmethod
308 def place_cars(list_of_cars):
309     cur_row = 7
310
311     for car in list_of_cars:
312         Application.cur_labels += 1
313         lb1 = tk.Label(text=f'{Application.cur_labels}', state='disabled', justify='center',
314                        disabledforeground='black',
315                        font='Arial 10', width=10, relief=tk.GROOVE)
316         lb1.grid(row=cur_row, column=0)
317         Application.labels.append(lb1)
318
319         lb2 = tk.Label(text=f'{car.type_car}', state='disabled', justify='center', disabledforeground='black',
320                        font='Arial 10', width=20, relief=tk.GROOVE)
321         lb2.grid(row=cur_row, column=1)
322         Application.labels.append(lb2)
323
324         lb3 = tk.Label(text=f'{car.capacity}', state='disabled', justify='center', disabledforeground='black',
325                        font='Arial 10', width=25, relief=tk.GROOVE)
326         lb3.grid(row=cur_row, column=2)
327         Application.labels.append(lb3)
328
329         lb4 = tk.Label(text=f'{car.length}', state='disabled', justify='center', disabledforeground='black',
330                        font='Arial 10', width=20, relief=tk.GROOVE)
331         lb4.grid(row=cur_row, column=3)
332         Application.labels.append(lb4)
333
334         lb5 = tk.Label(text=f'{car.width}', state='disabled', justify='center', disabledforeground='black',
335                        font='Arial 10', width=20, relief=tk.GROOVE)
336         lb5.grid(row=cur_row, column=4)
337         Application.labels.append(lb5)
338
339         lb6 = tk.Label(text=f'{car.height}', state='disabled', justify='center', disabledforeground='black',
340                        font='Arial 10', width=20, relief=tk.GROOVE)
341         lb6.grid(row=cur_row, column=5)
342         Application.labels.append(lb6)
343         if car.is_free == 'True':
344             lb7 = tk.Label(text='Свободна', state='disabled', justify='center', disabledforeground='black',
345                            font='Arial 10', width=20, relief=tk.GROOVE)
346         else:
347             lb7 = tk.Label(text='Занята', state='disabled', justify='center', disabledforeground='black',
348                            font='Arial 10', width=20, relief=tk.GROOVE)
349         lb7.grid(row=cur_row, column=6)
350         Application.labels.append(lb7)
351         cur_row += 1

```

Рисунок 8 – Метод place\_cars

После размещения информации об автомобилях окно программы выглядит следующим образом (см. Рисунок 9).

№	Тип	Грузоподъемность, тонн	Длина	Ширина	Высота	Статус
1	Бычок	3	4-2-5	2-2-2	2-2-4	Свободна
2	MAN-10	10	13.6	2.46	2.5-2.7	Свободна
3	Фура	10	2.3-2.8	2.5	2	Свободна

Рисунок 9 – Окно программы

За добавление транспорта отвечают метод add\_car и статический метод create\_car. В методе add\_car (см. Рисунок 10) создается отдельное окно, где

имеются поля ввода для нескольких характеристик: тип транспорта, грузоподъемность, длина, ширина, высота. Так же имеется кнопка “Применить”, при нажатии на нее вызывается метод `create_car`, в который передаются сами поля ввода (см. Рисунок 11).

```

280     def add_car(self):
281         add_settings = tk.Toplevel(self.window)
282         add_settings.wm_title('Добавление транспорта')
283
284         type_entry = tk.Entry(add_settings, justify=tk.RIGHT)
285         type_entry.grid(row=0, column=1, padx=20, pady=20)
286         tk.Label(add_settings, text='Тип транспорта').grid(row=0, column=0)
287
288         capacity_entry = tk.Entry(add_settings, justify=tk.RIGHT)
289         capacity_entry.grid(row=1, column=1, padx=20, pady=20)
290         tk.Label(add_settings, text='Грузоподъемность').grid(row=1, column=0)
291
292         length_entry = tk.Entry(add_settings, justify=tk.RIGHT)
293         length_entry.grid(row=2, column=1, padx=20, pady=20)
294         tk.Label(add_settings, text='Длина').grid(row=2, column=0)
295
296         width_entry = tk.Entry(add_settings, justify=tk.RIGHT)
297         width_entry.grid(row=3, column=1, padx=20, pady=20)
298         tk.Label(add_settings, text='Ширина').grid(row=3, column=0)
299
300         height_entry = tk.Entry(add_settings, justify=tk.RIGHT)
301         height_entry.grid(row=4, column=1, padx=20, pady=20)
302         tk.Label(add_settings, text='Высота').grid(row=4, column=0)
303         tk.Button(add_settings, text='Применить',
304                 command=lambda: self.create_car(type_entry, capacity_entry, length_entry, width_entry,
305                                                 height_entry)).grid(row=5, column=0, colspan=2, pady=20)

```

Рисунок 10 – Метод `add_car`

Рисунок 11 – Окно добавления транспорта

В методе `create_car` (см. Рисунок 12) значения из полей ввода сохраняются в переменные `new_length`, `new_width`, `new_height`, при этом разбиваясь по знаку ‘-’

(это сделано для того, чтобы пользователь мог вводить промежуток, а не конкретное значение). После этого значения проверяются на корректность ввода: если введенные характеристики нельзя представить в виде типа float (кроме типа ТС), выводится сообщения об ошибке с помощью `showerror` из библиотеки `tkinter`. После проверки создается новый экземпляр класса `Car`, куда записываются введенные данные, а также предварительно увеличивается на единицу количество “занятых” виджетами рядов (`Application.cur_labels`). По умолчанию новое транспортное средство является свободным. Далее полученная информация размещается в окне приложения (см. Рисунок 13) и происходит запрос к БД `Cars` для того, чтобы сохранить туда новую информацию (осуществляется с помощью `INSERT into`). Пользователю отображается уведомление (`showinfo`) о том, что добавление ТС прошло успешно. Завершается метод вызовом другого – `sorting`.

```

353 def create_car(self, type_car: tk.Entry, capacity: tk.Entry, length: tk.Entry, width: tk.Entry, height: tk.Entry):
354     new_length = length.get().split('-')
355     new_width = width.get().split('-')
356     new_height = height.get().split('-')
357     for i in new_length + new_width + new_height:
358         try:
359             float(i)
360         except ValueError:
361             showerror('Ошибка', 'Вы ввели неправильные значения')
362             return
363     try:
364         float(capacity.get())
365     except ValueError:
366         showerror('Ошибка', 'Вы ввели неправильные значения')
367         return
368     capacity = int(capacity.get())
369     length = length.get()
370     width = width.get()
371     height = height.get()
372     Application.cur_labels += 1
373     new_car = Car(type_car.get(), capacity, length, width, height, is_free='True')
374     Application.transport.append(new_car)
375
376     cur_row = 7 + Application.cur_labels
377
378     lb1 = tk.Label(text=f'{Application.cur_labels}', state='disabled', justify='center',
379                   disabledforeground='black', font='Arial 10', width=10, relief=tk.GROOVE)
380     lb1.grid(row=cur_row, column=0)
381     Application.labels.append(lb1)
382
383     lb2 = tk.Label(text=f'{type_car.get()}', state='disabled', justify='center',
384                   disabledforeground='black', font='Arial 10', width=20, relief=tk.GROOVE)
385     lb2.grid(row=cur_row, column=1)
386     Application.labels.append(lb2)
387
388     lb3 = tk.Label(text=f'{capacity}', state='disabled', justify='center', disabledforeground='black',
389                   font='Arial 10', width=25, relief=tk.GROOVE)
390     lb3.grid(row=cur_row, column=2)
391     Application.labels.append(lb3)
392
393     lb4 = tk.Label(text=f'{length}', state='disabled', justify='center', disabledforeground='black',
394                   font='Arial 10', width=20, relief=tk.GROOVE)
395     lb4.grid(row=cur_row, column=3)
396     Application.labels.append(lb4)
397
398     lb5 = tk.Label(text=f'{width}', state='disabled', justify='center', disabledforeground='black',
399                   font='Arial 10', width=20, relief=tk.GROOVE)
400     lb5.grid(row=cur_row, column=4)
401     Application.labels.append(lb5)
402
403     lb6 = tk.Label(text=f'{height}', state='disabled', justify='center', disabledforeground='black',
404                   font='Arial 10', width=20, relief=tk.GROOVE)
405     lb6.grid(row=cur_row, column=5)
406     Application.labels.append(lb6)
407
408     lb7 = tk.Label(text='Свободна', state='disabled', justify='center', disabledforeground='black',
409                   font='Arial 10', width=20, relief=tk.GROOVE)
410     lb7.grid(row=cur_row, column=6)
411     Application.labels.append(lb7)
412
413     db = sqlite3.connect('Cars.db')
414     c = db.cursor()
415     c.execute(
416         f"INSERT INTO cars VALUES ({new_car.type_car}, {new_car.capacity}, {new_car.length}, \
417         {new_car.width}, {new_car.height}, 'True')"
418     )
419     db.commit()
420     db.close()
421     showinfo('Добавление', 'ТС успешно добавлено')
422     return self.sorting()

```

Рисунок 12 – Метод create\_car

Программное обеспечение

Создать заказ

Удалить заказ

Добавить транспорт

Удалить транспорт

№	Тип	Грузоподъемность, тонн	Длина	Ширина	Высота	Статус
1	Бычок	3	4.2-5	2.2-2	2.2-4	Свободна
2	MAN-10	10	13.6	2.46	2.5-2.7	Свободна
3	Фура	10	2.3-2.8	2.5	2	Свободна
4	HUMMER	3	2.5-3	2.2-2.3	3.1	Свободна

Добавление

Весь транспорт

Свободный транспорт

Занятый транспорт

По грузоподъемности

ТС успешно добавлено

OK

Рисунок 13 – Результат добавления транспорта

12

Метод `sorting` привязан к экземплярам класса `Radiobutton`, размещенных в главном окне (их размещение происходит в методе `create_widgets`). Перед определением классов был создан словарь `view_mode` (см. Рисунок 14), в котором ключами являются номера 1-4, а значениями – режимы просмотра пользователем ТС. Текущий режим просмотра (вернее, его ключ), хранится в переменной `Application.view` (по умолчанию – 1). В методе `sorting` (см. Рисунок 15) определяется размещение виджетов в зависимости от выбранного режима.

```

5  view_mode = {
6      1: 'Весь транспорт',
7      2: 'Свободный транспорт',
8      3: 'Занятый транспорт',
9      4: 'По грузоподъемности'
10 }

```

Рисунок 14 – Словарь `view_mode`

```

206 def sorting(self):
207     view_var = int(Application.view.get())
208     if view_var == 1:
209         return self.refresh()
210     elif view_var == 2:
211         for i in Application.labels:
212             i.destroy()
213         new_cars = []
214         for i in Application.transport:
215             if i.is_free == 'True':
216                 new_cars.append(i)
217         Application.labels.clear()
218         Application.cur_labels = 0
219         return self.place_cars(new_cars)
220     elif view_var == 3:
221         for i in Application.labels:
222             i.destroy()
223         new_cars = []
224         for i in Application.transport:
225             if i.is_free == 'False':
226                 new_cars.append(i)
227         Application.labels.clear()
228         Application.cur_labels = 0
229         return self.place_cars(new_cars)
230     elif view_var == 4:
231         for i in Application.labels:
232             i.destroy()
233         Application.labels.clear()
234         Application.cur_labels = 0
235         Application.transport.sort(key=lambda x: x.capacity)
236         Application.transport.reverse()
237         return self.place_cars(Application.transport)

```

Рисунок 15 – Метод `sorting`

Если выбран режим “Весь транспорт”, вызывается метод `refresh` (см. Рисунок 16), в котором все лейблы, хранящиеся в списке `Application.labels` удаляются, списки `Application.labels` и `Application.transport` очищаются и вновь вызывается метод `create_widgets`.

Если выбран режим “Свободный транспорт”, вся информация об автомобилях также удаляется с главного окна, и формируется новый список `new_cars`, в который добавляются только те элементы списка `Application.transport`, у которых



значение атрибута `is_free` принимает значение “True”. После чего вызывается метод `place_cars`, в который передается новый список.

Если выбран режим “Занятый транспорт”, аналогично формируется список с занятыми машинами.

При выборе режима “По грузоподъемности” все лейблы также уничтожаются, список `Application.transport` сортируется с помощью лямбда функции по грузоподъемности элементов. Список реверсируется, и осуществляется вызов метода `place_cars`, который будет располагать ТС по возрастанию грузоподъемности.

```
423     def refresh(self):
424         for i in Application.labels:
425             i.destroy()
426         Application.labels.clear()
427         Application.transport.clear()
428         Application.cur_labels = 0
429         return self.initialization()
```

Рисунок 16 – Метод `refresh`

За удаление транспортного средства отвечают методы `remove_car` и `delete_car`. В методе `remove_car` (см. Рисунок 17) выполняется проверка выбранного режима – удалять автомобиль можно только в режиме “Весь транспорт”. Создается новое окно, в котором лишь одно поле ввода – номер транспортного средства, которое пользователь хочет удалить. Кнопка “Удалить” вызывает метод `delete_car`, куда передается поле ввода `type_entry`.

```
239     def remove_car(self):
240         if self.view.get() != 1:
241             showerror('Ошибка', 'Удалить ТС можно только в режиме "Весь транспорт"')
242             return
243         remove_settings = tk.Toplevel(self.window)
244         remove_settings.wm_title('Удаление транспорта')
245
246         type_entry = tk.Entry(remove_settings, justify=tk.RIGHT)
247         type_entry.grid(row=0, column=1, padx=20, pady=20)
248         tk.Label(remove_settings, text='Введите номер ТС, которое хотите удалить').grid(row=0, column=0)
249         tk.Button(remove_settings, text='Удалить', command=lambda: self.delete_car(type_entry)).grid(row=1, column=1,
250                                                                                               pady=20)
```

Рисунок 17 – Метод `remove_car`

В методе `delete_car` (см. Рисунок 18) выполняется проверка введенного значения: является ли оно целочисленным и имеется ли автомобиль с таким номером. В противном случае на экран выводится ошибка. Характеристики выбранного автомобиля сохраняются в соответствующие переменные, после чего происходит запрос к базе данных и удаление записи (с помощью `DELETE FROM`)

о ТС. После этого вызывается метод `refresh`, благодаря которому информация об оставшихся автомобилях отобразится без ошибок.

```
def delete_car(self, number: tk.Entry):
    try:
        int(number.get())
    except ValueError:
        showerror('Ошибка', 'Вы ввели неправильное значение')
        return

    deleted_number = int(number.get())
    try:
        Application.transport[deleted_number - 1]
    except IndexError:
        showerror('Ошибка', 'Вы ввели неправильное значение')
        return

    deleted_car = Application.transport[deleted_number - 1]
    type_car = deleted_car.type_car
    capacity = deleted_car.capacity
    length = deleted_car.length
    width = deleted_car.width
    height = deleted_car.height
    db = sqlite3.connect('Cars.db')
    cursor = db.cursor()
    cursor.execute(
        f"DELETE FROM cars WHERE type = '{type_car}' AND capacity = {capacity} AND length = '{length}' \
        AND width = '{width}' AND height = '{height}'")
    db.commit()
    db.close()
    return self.refresh()
```

Рисунок 18 – Метод `delete_car`

На создание заказа также отведено 2 метода: `create_order` и `make_order`. В методе `create_order` (см. Рисунок 19) создается новое окно с несколькими полями ввода: вес, длина, ширина, высота (см. Рисунок 20). Они нужны для ввода характеристик груза пользователя, чтобы впоследствии подобрать машину для перевозки. Кнопка “Создать заказ” вызывает метод `make_order`, передавая все поля ввода.

```
def create_order(self):
    order_settings = tk.Toplevel(self.window)
    order_settings.wm_title('Создание заказа')

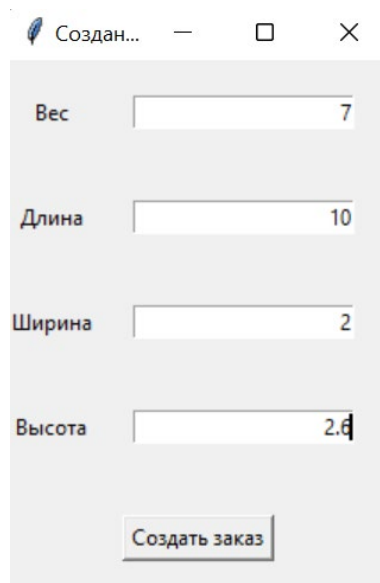
    capacity_entry = tk.Entry(order_settings, justify=tk.RIGHT)
    capacity_entry.grid(row=0, column=1, padx=20, pady=20)
    tk.Label(order_settings, text='Вес').grid(row=0, column=0)

    length_entry = tk.Entry(order_settings, justify=tk.RIGHT)
    length_entry.grid(row=1, column=1, padx=20, pady=20)
    tk.Label(order_settings, text='Длина').grid(row=1, column=0)

    width_entry = tk.Entry(order_settings, justify=tk.RIGHT)
    width_entry.grid(row=2, column=1, padx=20, pady=20)
    tk.Label(order_settings, text='Ширина').grid(row=2, column=0)

    height_entry = tk.Entry(order_settings, justify=tk.RIGHT)
    height_entry.grid(row=3, column=1, padx=20, pady=20)
    tk.Label(order_settings, text='Высота').grid(row=3, column=0)
    tk.Button(order_settings, text='Создать заказ',
              command=lambda: self.make_order(capacity_entry, length_entry, width_entry,
                                              height_entry)).grid(row=4, column=0, columnspan=2, pady=20)
```

Рисунок 19 – Метод `create_order`



Создан...

Вес 7

Длина 10

Ширина 2

Высота 2.6

Создать заказ

Рисунок 20 – Окно создания заказа

Метод `make_order` является статическим (см. Рисунок 21). Введенные значения проверяются на корректность – все они должны являться числами. После выполняется цикл по всему транспорту: некоторые характеристики ТС могут быть представлены в виде промежутка, поэтому для каждого автомобиля определяется минимальная/максимальная длина/ширина/высота, которую он может перевезти (если характеристика представлена единственным значением, а не промежутком, минимальное значение принимается за 0). Выполняется проверка, свободен ли данный транспорт и сможет ли он доставить груз. При выполнении всех условий происходит подключение к БД, обновляется информация в ячейке `is_free` для текущей машины – ТС становится занято. Изменения происходят также в атрибуте `is_free` экземпляра подходящего автомобиля, текст на лейбле изменяется на “Занято”. На экран выводится сообщение об успешном оформлении заказа и номер ТС, которое осуществит перевозку. В противном случае выводится ошибка.



```

104 @staticmethod
105 def make_order(capacity: tk.Entry, length: tk.Entry, width: tk.Entry, height: tk.Entry):
106     for i in [capacity.get(), length.get(), width.get(), height.get()]:
107         try:
108             float(i)
109         except ValueError:
110             showerror('Ошибка', 'Вы ввели неправильное значение')
111             return
112     capacity = float(capacity.get())
113     length = float(length.get())
114     width = float(width.get())
115     height = float(height.get())
116     cur_car = 0
117     for car in Application.transport:
118         cur_car += 1
119         diap_len = car.length.split('-')
120         diap_wid = car.width.split('-')
121         diap_he = car.height.split('-')
122
123         if car.is_free == 'False':
124             continue
125
126         if len(diap_len) == 2:
127             max_len = float(diap_len[1])
128             min_len = float(diap_len[0])
129         else:
130             max_len = float(diap_len[0])
131             min_len = 0
132
133         if len(diap_wid) == 2:
134             max_wid = float(diap_wid[1])
135             min_wid = float(diap_wid[0])
136         else:
137             max_wid = float(diap_wid[0])
138             min_wid = 0
139
140         if len(diap_he) == 2:
141             max_he = float(diap_he[1])
142             min_he = float(diap_he[0])
143         else:
144             max_he = float(diap_he[0])
145             min_he = 0
146         if capacity <= float(car.capacity) and min_len <= length <= max_len and min_wid <= width <= max_wid \
147             and min_he <= height <= max_he:
148             db = sqlite3.connect('Cars.db')
149             cursor = db.cursor()
150             cursor.execute(
151                 f"UPDATE cars SET is_free = 'False' WHERE type = '{car.type_car}' AND capacity = {car.capacity} \
152                     AND length = '{car.length}' AND width = '{car.width}' AND height = '{car.height}'")
153             db.commit()
154             db.close()
155             car.is_free = 'False'
156             Application.labels[cur_car * 7 - 1]['text'] = 'Занята'
157             return showinfo('Успешно', f'Заказ успешно добавлен! ТС: {car.type_car}')
158
159     return showinfo('Не успешно', f'Нет подходящей машины для вашего заказа')

```

Рисунок 21 – Метод make\_order

Последняя функция ПО – удаление заказа, осуществляется методами remove\_order и delete\_order. Метод remove\_order (см. Рисунок 22) реализован так же, как и метод remove\_car: создается окно, в котором единственное поле ввода – номер машины, у которой хотят снять заказ. Данное поле ввода передается в метод delete\_order.

```

def remove_order(self):
    if self.view.get() != 1:
        showerror('Ошибка', 'Удалить заказ можно только в режиме "Весь транспорт"')
        return
    remove_settings = tk.Toplevel(self.window)
    remove_settings.wm_title('Удаление заказа')

    type_entry = tk.Entry(remove_settings, justify=tk.RIGHT)
    type_entry.grid(row=0, column=1, padx=20, pady=20)
    tk.Label(remove_settings, text='Введите номер ТС, с которого хотите снять заказ').grid(row=0, column=0)
    tk.Button(remove_settings, text='Удалить заказ', command=lambda: self.delete_order(type_entry)).grid(row=1,
                                                                                                     column=1,
                                                                                                     pady=20)

```

Рисунок 22 – Метод remove\_order

Метод delete\_order (см. Рисунок 23) выполняет проверку на корректность ввода: введенные значения должны быть целочисленными и не превышать наибольший номер среди ТС. После этого рассмотрено два случая: если выбранная машина свободна – выводится ошибка с информацией об этом, если текущее ТС имеет заказ, осуществляется запрос в БД, обновляется информация в столбце is\_free, изменяются атрибут is\_free экземпляра класса Car и текст лейбла. Вызывается метод refresh.

```

def delete_order(self, number: tk.Entry):
    try:
        int(number.get())
    except ValueError:
        showerror('Ошибка', 'Вы ввели неправильное значение')
        return

    deleted_number = int(number.get())
    try:
        Application.transport[deleted_number - 1]
    except IndexError:
        showerror('Ошибка', 'Вы ввели неправильное значение')
        return

    deleted_order_car = Application.transport[deleted_number - 1]
    if deleted_order_car.is_free == 'True':
        return showerror('Ошибка', 'Текущее ТС не имеет заказов')
    deleted_order_car.is_free = 'True'
    Application.labels[deleted_number * 7 - 1]['text'] = 'Свободна'
    type_car = deleted_order_car.type_car
    capacity = deleted_order_car.capacity
    length = deleted_order_car.length
    width = deleted_order_car.width
    height = deleted_order_car.height
    db = sqlite3.connect('Cars.db')
    cursor = db.cursor()
    cursor.execute(f"UPDATE cars SET is_free = 'True' WHERE type = '{type_car}' AND capacity = '{capacity}' \
AND length = '{length}' AND width = '{width}' AND height = '{height}'")
    db.commit()
    db.close()
    return self.refresh()

```

Рисунок 23 – Метод delete\_order

## ЗАКЛЮЧЕНИЕ

В процессе работы было реализовано ПО учета грузового транспорта для Автотранспортного отдела логистической компании. Были выполнены все поставленные задачи, программа представляет собой завершенное приложение. Возможно сохранение данных в базу данных. Графический интерфейс реализован с помощью библиотеки tkinter.