МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ст. преподаватель |  |  |  | Е. О. Шумова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ОРГАНИЗАЦИИ ВЗАИМОДЕЙСТВИЯ ОБЪЕКТОВ ПРИ ЗАДАННЫХ КРИТЕРИЯХ |
| по дисциплине: ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4134К |  |  |  | Д. В. Самарин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

Оглавление

[1. Введение 3](#_Toc153973956)

[2. Постановка задачи 3](#_Toc153973957)

[2.1. Анализ предметной области 3](#_Toc153973958)

[2.2. Формулировка технического задания 4](#_Toc153973959)

[3. Разработка классов 5](#_Toc153973960)

[3.1. Классы сущностей: 6](#_Toc153973961)

[3.2. Управляющие классы: 8](#_Toc153973962)

[3.3. Интерфейсные классы: 8](#_Toc153973963)

[3.4. Используемые паттерны проектирования (их обоснования): 9](#_Toc153973964)

[3.5. Диаграмма классов: 9](#_Toc153973965)

[4. Разработка приложения 12](#_Toc153973966)

[5. Тестирование 25](#_Toc153973967)

[6. Приложение с полным кодом программ 37](#_Toc153973968)

[7. Заключение 63](#_Toc153973969)

[8. Список использованных источников 63](#_Toc153973970)

# Введение

В ходе курсового проекта необходимо разработать приложение,

позволяющее протестировать взаимодействие объектов классов,

спроектированных и реализованных мной для решения конкретной

задачи при заданных критериях.

При разработке программного обеспечения следует использовать

шаблоны проектирования. Также предпочтение должно быть отдано

графическому приложению.

Целями курсового проекта являются:

- закрепление знаний, полученных в ходе изучения дисциплины

«Объектно-ориентированное программирование»;

- приобретение навыков практического программирования с

использованием объектно-ориентированной парадигмы;

- подготовка к выполнению выпускной квалификационной работы.

Задачей курсового проекта является разработка иерархии типов в

заданной предметной области, включающей в себя:

- разработку иерархии классов

- разработку приложения

- подготовку презентации.

Актуальность проекта на высоком уровне. В наше время существует множество дилерских центров. Подобные приложения делают жизнь подобных организаций проще.

Ниже будет расписана основная часть пояснительной записки, которая включается в себя: постановку задачи, разработку классов, разработку приложения, тестирование.

# Постановка задачи

Данная курсовая работа по программированию направлена на разработку системы классов, которая будет описывать различные транспортные средства, управляемые дилерским центром. Основной целью проектирования является создание программного продукта, который позволит учитывать закупки различных брендов транспортных средств от производителей, а также фиксировать данные о продажах и покупателях.

## 2.1. Анализ предметной области

Предметной областью является управление транспортными средствами в дилерском центре. Основными сущностями предметной области являются транспортные средства (автомобили, мотоциклы и другие), производители, дилерский центр, покупатели, операции закупки и продажи.

Словарь предметной области:

- Транспортное средство

- Производитель

- Дилерский центр

- Покупатель

- Закупка

- Продажа

Функциональные требования:

- Ведение данных о закупках транспортных средств от различных производителей.

- Фиксация данных о продажах транспортных средств покупателям.

- Сохранение информации о производителях, моделях, характеристиках транспортных средств.

- Учет данных о покупателях, их контактных данных.

## 2.2. Формулировка технического задания

Наименование проекта: разработка системы классов для учета транспортных средств в дилерском центре.

Задача проекта:

Разработать систему классов, предназначенную для учета и управления данными о различных транспортных средствах, которые закупаются дилерским центром от различных производителей. Система должна обеспечивать учет процессов закупки, продажи транспортных средств, а также хранение данных о покупателях.

Описание проекта:

Необходимо создать программную систему, которая позволит дилерскому центру вести учет закупок различных брендов транспортных средств у производителей и осуществлять отслеживание продаж этих транспортных средств конечным покупателям. Система должна предоставлять следующие функциональные возможности:

Учет закупок:

Регистрация данных о транспортных средствах от производителей.

Хранение информации о поставщиках и поставках, включая даты, количество и характеристики транспортных средств.

Учет продаж:

Фиксация данных о продажах транспортных средств конечным покупателям.

Отслеживание информации о покупателях, включая их контактные данные.

Управление данными о транспортных средствах:

Хранение характеристик каждого транспортного средства, включая модель, год выпуска, цену и другие характеристики.

Возможность редактирования и обновления данных о транспортных средствах.

Технические требования:

Разработка системы должна быть выполнена на QTCreator.

Система должна быть разделена на классы, обеспечивающие модульность и расширяемость.

Интерфейс пользователя должен быть интуитивно понятным и обеспечивать доступ к функциональным возможностям системы.

Итогом проделанных работ станет: разработка и внедрение системы классов для учета транспортных средств, обеспечивающей эффективное управление данными о закупках, продажах и клиентах дилерского центра.

# 3. Разработка классов

Для разработки иерархии классов мы начнем с выделения основных сущностей предметной области и определения классов, описывающих эти сущности. Затем мы определим управляющие классы и интерфейсные классы для организации взаимодействия между ними и с внешней средой. Ниже приведена детальная разработка иерархии классов.

## 3.1. Классы сущностей:

1. Транспортное средство (Vehicle):

- Поля:

- ID

- Марка

- Модель

- Год выпуска

- Стоимость

- Описание

- Статус (доступно, продано, зарезервировано и т.д.)

- Методы:

- Конструктор для создания нового транспортного средства

- Методы для получения и установки значений полей

- Метод для изменения статуса

2. Производитель (Manufacturer):

- Поля:

- ID

- Название

- Страна происхождения

- Методы:

- Конструктор для создания нового производителя

- Методы для получения и установки значений полей

3. Дилерский центр (Dealership):

- Поля:

- ID

- Название

- Адрес

- Список доступных транспортных средств

- Методы:

- Конструктор для создания нового дилерского центра

- Методы для добавления и удаления транспортных средств

- Методы для получения информации о транспортных средствах в наличии

4. Покупатель (Customer):

- Поля:

- ID

- Имя

- Фамилия

- Контактная информация

- Методы:

- Конструктор для создания нового покупателя

- Методы для получения и установки значений полей

5. Операция продажи (Sale):

- Поля:

- ID

- Дата продажи

- Продавец (сотрудник дилерского центра)

- Покупатель

- Транспортное средство

- Сумма продажи

- Методы:

- Конструктор для создания новой операции продажи

- Методы для получения и установки значений полей

## 3.2. Управляющие классы:

1. Склад (Inventory):

- Поля:

- Список всех доступных транспортных средств

- Методы:

- Методы для добавления и удаления транспортных средств

- Методы для поиска транспортных средств по различным критериям (марка, модель, статус и т.д.)

2. Продажи (SalesManager):

- Поля:

- Список всех операций продажи

- Методы:

- Методы для регистрации новой операции продажи

- Методы для отчетности по продажам

## 3.3. Интерфейсные классы:

1. Графический интерфейс (GUI):

- Методы:

- Методы для взаимодействия с пользователем, включая ввод и вывод данных.

Обоснования применения классов в нашей будущей программе:

- Использование классов сущностей для описания базовых объектов предметной области позволяет удобно хранить и обрабатывать информацию о транспортных средствах, производителях, дилерских центрах, покупателях и операциях продажи.

- Управляющие классы, такие как Склад и Продажи, позволяют организовать управление данными и операциями, связанными с транспортными средствами и продажами.

- Интерфейсный класс GUI обеспечивает взаимодействие пользователя с программой через графический интерфейс, что делает использование приложения более удобным для конечных пользователей.

## 3.4. Используемые паттерны проектирования (их обоснования):

1. Фабричный метод (Factory Method): Использование конструкторов для создания диалоговых окон (**VehicleDialog**, **ManufacturerDialog**, и т. д.).

2. Одиночка (Singleton): Паттерн Singleton будет использован для создания и управления объектом базы данных или же для управления доступом к хранилищу данных о транспортных средствах и продажах. Это позволит обеспечить единственный экземпляр класса для доступа к данным из разных частей приложения.

3. Стратегия (Strategy): В методе **updateTable()**, в зависимости от выбранной вкладки в **tabWidget**, различные таблицы обновляются разными способами.

4. Model-View-Controller (MVC): Код будет организован в соответствии с паттерном MVC. Будет присутствовать **MainWindow**, который представляет пользовательский интерфейс (**View**), использует модель данных (**model**) и обрабатывает события (слоты) как контроллер.

## 3.5. Диаграмма классов:

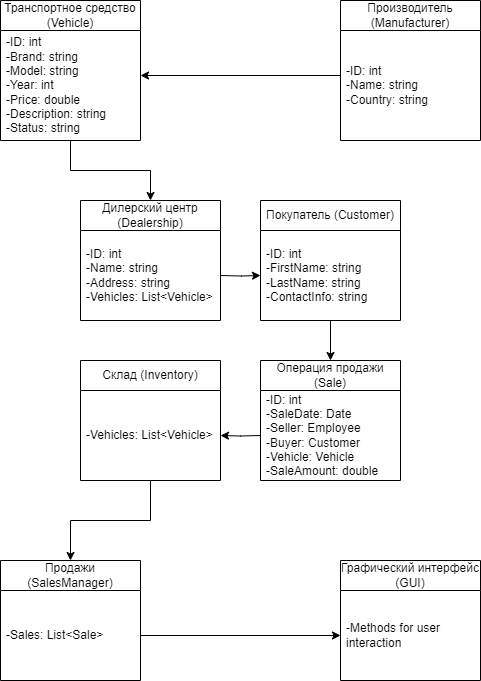


Рисунок 1 – диаграмма классов

Помимо этого, рассмотрим диаграммы для наших будущих паттернов:

Factory Method:

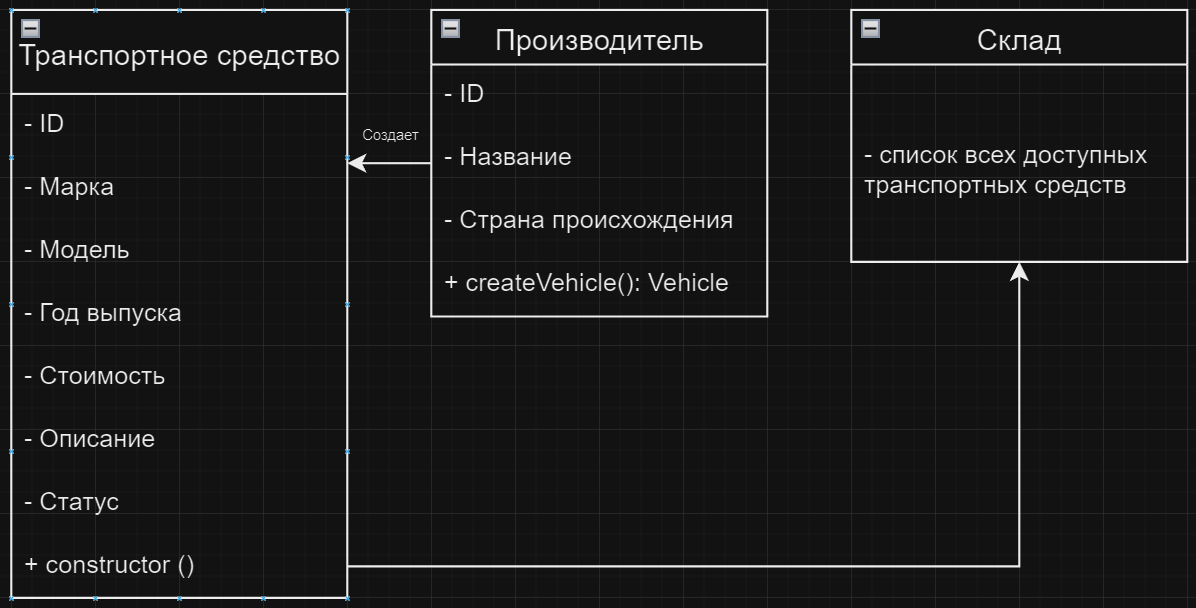


Рисунок 2 – диаграмма FM

Singleton:

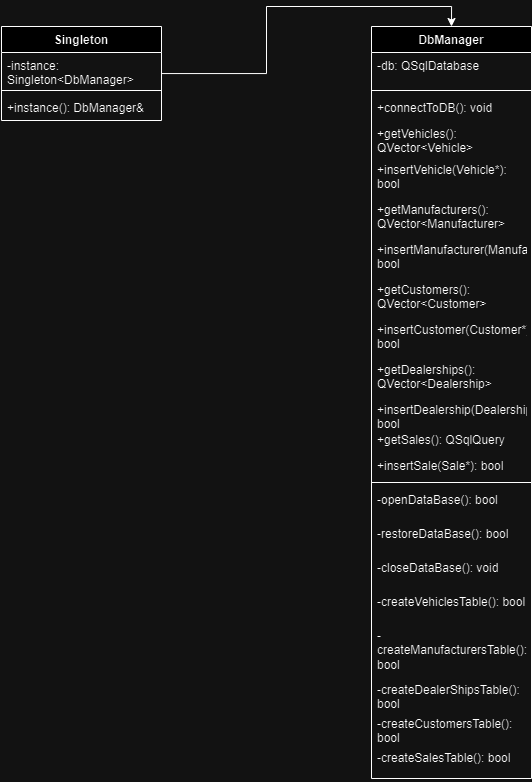


Рисунок 3 – диаграмма Singleton

Strategy:

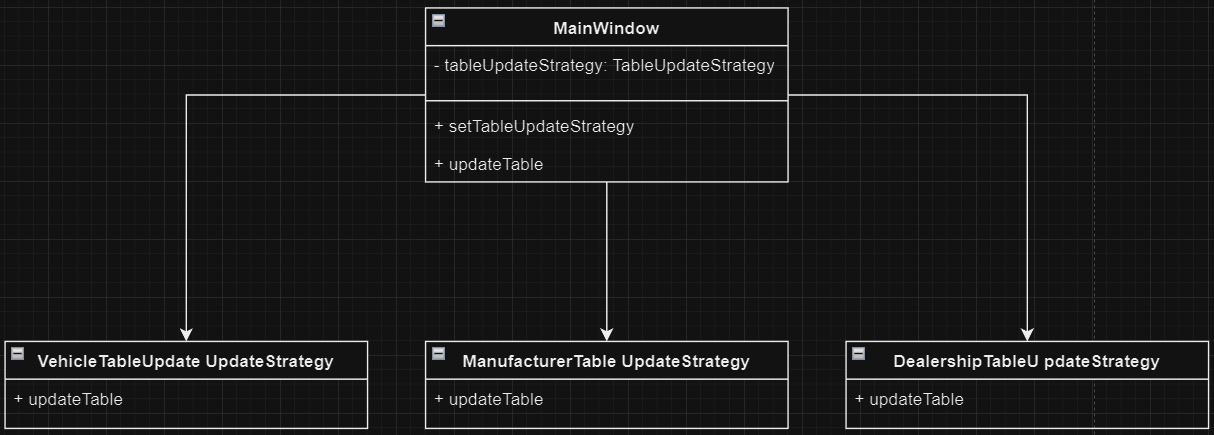


Рисунок 4 – диаграмма Strategy

Model-View-Controller (MVC):

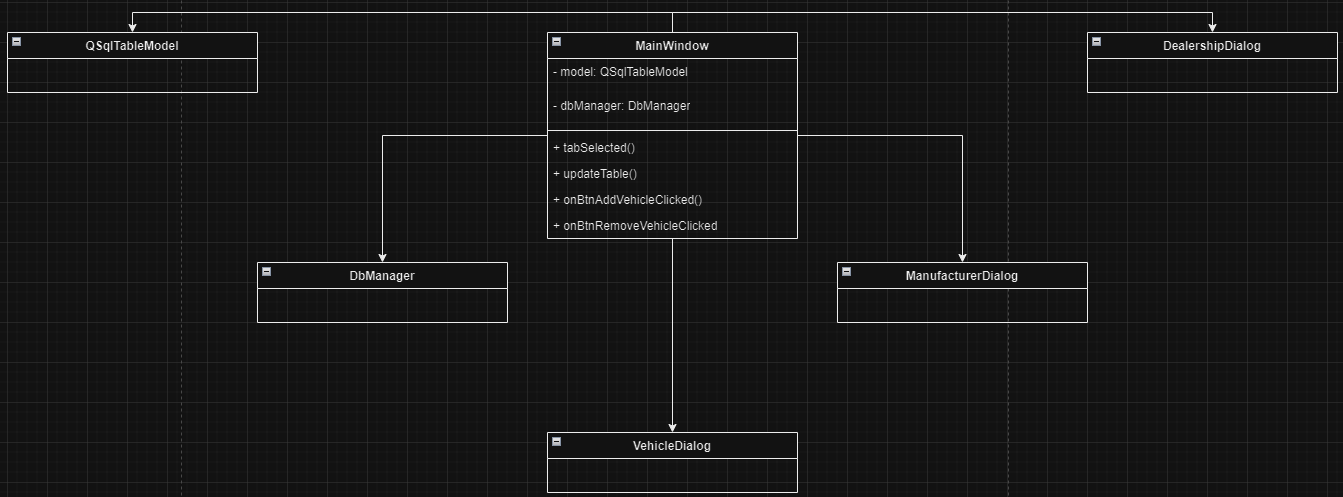


Рисунок 5 – диаграмма MVC

Таким образом созданная иерархия классов и применяемые паттерны позволят эффективно управлять данными и операциями, связанными с транспортными средствами и продажами в дилерском центре, а также обеспечит удобный интерфейс для пользователей.

# 4. Разработка приложения

**4.1. Разработка интерфейса приложения**

Приложение представляет собой удобную систему управления продажами транспортных средств. Интерфейс прост и интуитивно понятен. Применяются различные элементы интерфейса, такие как: QTable, PushButton и д.р.

Работа с БД осуществляется через добавление данных в открывающихся формах ui. Каждое поле данных подписано, пользователю не составит труда заполнить информацию с нуля.

Применяются приятные для восприятия глазу цвета интерфейса.

Рассмотрим структуру пользовательского интерфейса:

Главный класс: QcentralWidget

Объекты:

1. tabWidget (QTabWidget)

1.1. tab (QWidget):

* btnAddVehicle (QPushButton)
* btnRemoveVehicle (QPushButton)
* tableVehicle (QTableView)

1.2. tab\_2 (QWidget):

* btnAddManufacturer (QPushButton)
* btnRemoveManufacturer (QPushButton)
* tableManufacturers (QTableView)

1.3. tab\_3 (QWidget):

* btnAddDealership (QPushButton)
* btnRemoveDealership (QPushButton)
* tableDealerships (QTableView)

1.4. tab\_4 (QWidget):

* btnAddCustomer (QPushButton)
* btnRemoveCustomer (QPushButton)
* tableCustomers (QTableView)

1.5. tab\_5 (QWidget):

* btnAddSale (QPushButton)
* btnRemoveSale (QPushButton)
* tableSales (QTableView)

А также диалоговые окна, вызываемые в ходе работы с программой:

1. CustomerDialog (QDialog)

1.1. Group1 (QGroupBox)

* btnCancel (QPushButton)
* btnOk (QPushButton)
* label (QLabel)
* label\_2 (QLabel)
* label\_3 (QLabel)
* txtContacts (QLineEdit)
* txtFName (QLineEdit)
* txtLName (QLineEdit)

2. DealershipDialog (QDialog)

2.1. Group2 (QGroupBox)

* btnCancel (QPushButton)
* btnOk (QPushButton)
* label (QLabel)
* label\_2 (QLabel)
* txtAddress (QLineEdit)
* txtName (QLineEdit)

3. ManufacturerDialog (QDialog)

3.1. Group3 (QGroupBox)

* btnCancel (QPushButton)
* btnOk (QPushButton)
* label (QLabel)
* label\_2 (QLabel)
* txtCountry (QLineEdit)
* txtName (QLineEdit)

4. SaleDialog (QDialog)

4.1. Group4 (QGroupBox)

* btnCancel (QPushButton)
* btnOk (QPushButton)
* cbCustomer (QComboBox)
* cbVehicle (QComboBox)
* dateEdit (QDateEdit)
* label (QLabel)
* label\_2 (QLabel)
* label\_3 (QLabel)
* label\_4 (QLabel)
* label\_5 (QLabel)
* txtAmount (QLineEdit)
* txtDealer (QLineEdit)

5. VehicleDialog (QDialog)

5.1. Group5 (QGroupBox)

* btnCancel (QPushButton)
* btnOk (QPushButton)
* label (QLabel)
* label\_2 (QLabel)
* label\_3 (QLabel)
* label\_4 (QLabel)
* label\_5 (QLabel)
* label\_6 (QLabel)
* txtDescription (QLineEdit)
* txtMark (QLineEdit)
* txtModel (QLineEdit)
* txtPrice (QLineEdit)
* txtStatus (QLineEdit)
* txtYear (QLineEdit)

Подробный разбор:

tab (QWidget):

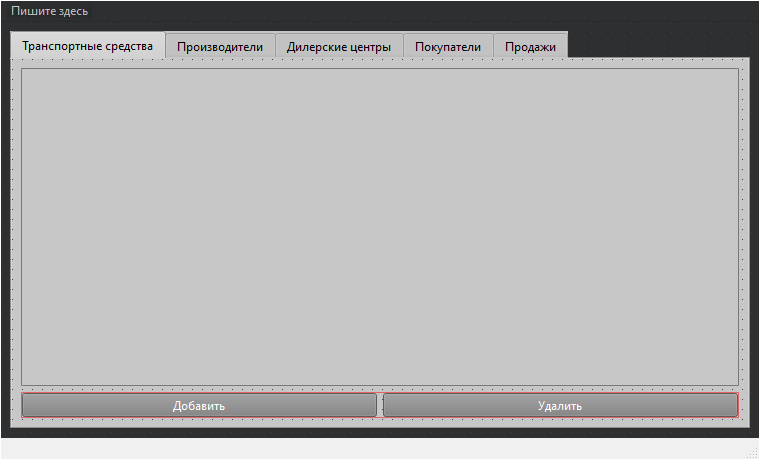


Рисунок 6 - tab (QWidget)

1. btnAddVehicle (QPushButton) – позволяет добавить новое транспортное средство. При нажатии выводится диалоговое окно VehicleDialog для ввода данных об авто.
2. btnRemoveVehicle (QPushButton) – позволяет удалить выбранное транспортное средство.
3. tableVehicle (QTableView) – отображает содержимое базы данных с хранимыми данными о транспорте.

tab\_2 (QWidget):

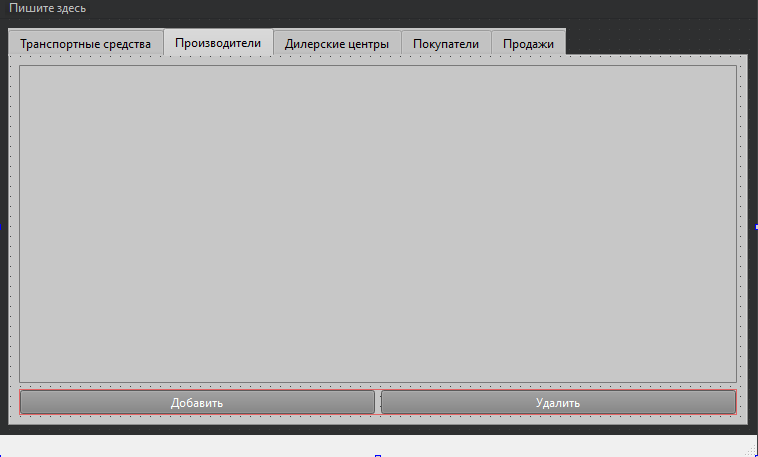


Рисунок 7 - tab\_2 (QWidget)

1. btnAddManufacturer (QPushButton) – позволяет добавить нового производителя. При нажатии выводит диалоговое окно ManufacturerDialog для ввода данных в БД.
2. btnRemoveManufacturer (QPushButton) – позволяет удалить выбранного производителя.
3. tableManufacturers (QTableView) – выводит таблицу с данными о производителя из БД. Позволяет работать с ними и вносить изменения.

tab\_2 (QWidget):

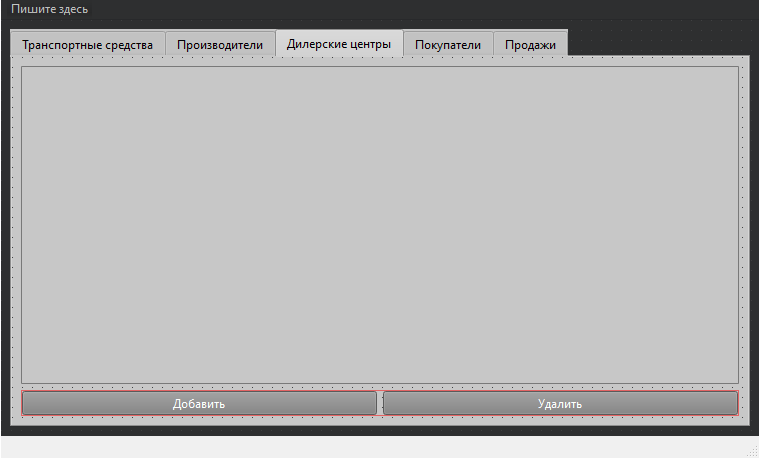


Рисунок 8 – tab\_3 (QWidget)

1. btnAddDealership (QPushButton) – позволяет добавить дилерский центр. При нажатии выводится диалоговое окно DealershipDialog позволяющее внести новые данные в БД.
2. btnRemoveDealership (QPushButton) – позволяет удалить выбранного дилера.
3. tableDealerships (QTableView) – выводит информацию о дилерах из базы данных, позволяет вносить изменения.

tab\_4 (QWidget):

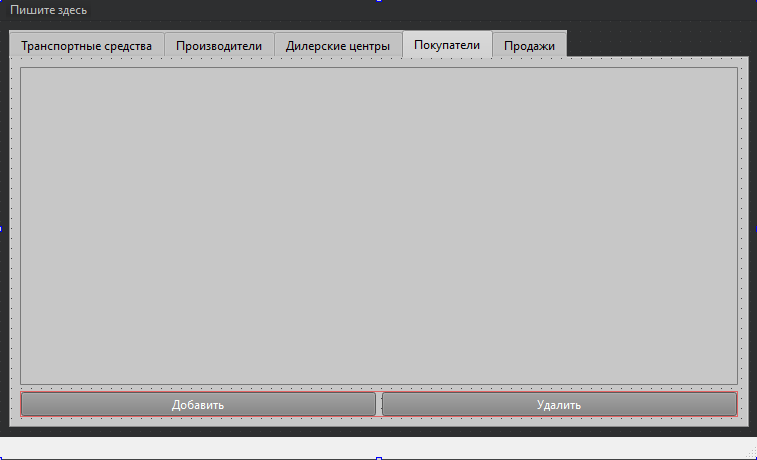


Рисунок 9 – tab\_4 (QWidget)

1. btnAddCustomer (QPushButton) – позволяет добавить нового покупателя в базу данных. При нажатии на кнопку выводится диалоговое окно CustomerDialog для внесения новых данных.
2. btnRemoveCustomer (QPushButton) – позволяет удалить выбранного покупателя из БД.
3. tableCustomers (QTableView) – выводит БД для последующей работы с ней. Возможно внесение изменений.

tab\_5 (QWidget):

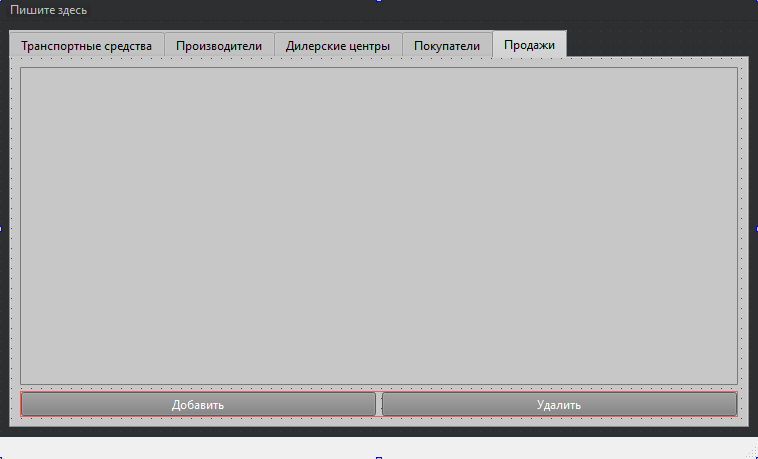


Рисунок 10 – tab\_5 (QWidget)

1. btnAddSale (QPushButton) – позволяет внести в базу данных информацию о проданном товаре. При нажатии на кнопку выводится диалоговое окно SaleDialog.
2. btnRemoveSale (QPushButton) – позволяет удалить из базы данных информацию о проданном товаре.
3. tableSales (QTableView) – выводит информацию о проданных товарах из базы данных

Рассмотрим диалоговые окна:

VehicleDialog (QDialog):

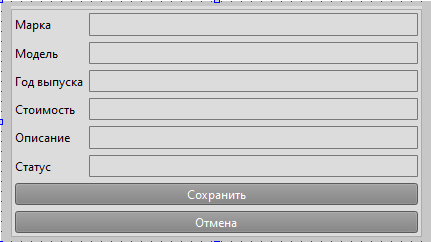


Рисунок 11 – VehicleDialog (QDialog)

1. btnCancel (QPushButton) – позволяет отменить операцию добавления нового транспортного средства.
2. btnOk (QPushButton) – вносит введенные изменения в БД.
3. label (QLabel) label\_2 (QLabel) label\_3 (QLabel) label\_4 (QLabel) label\_5 (QLabel) label\_6 (QLabel) – текстовая информация для пользователя. Даёт понимание пользователю о полях ввода (какую информацию вводить).
4. txtDescription (QLineEdit) – поле для ввода описания транспорта.
5. txtMark (QLineEdit) – поле для ввода марки авто.
6. txtModel (QLineEdit) – поле для ввода модели авто.
7. txtPrice (QLineEdit) – поле для ввода стоимости авто.
8. txtStatus (QLineEdit) – поле для ввода статуса авто.
9. txtYear (QLineEdit) – поле для ввода года авто.

ManufacturerDialog (QDialog):

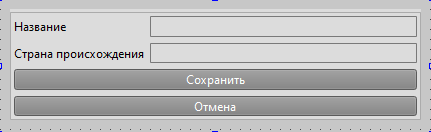


Рисунок 12 – ManufacturerDialog (QDialog)

1. btnCancel (QPushButton) – позволяет отменить операцию добавления нового транспортного средства.
2. btnOk (QPushButton) – вносит введенные изменения в БД.
3. label (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Название производителя.
4. label\_2 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Страна происхождения.
5. txtCountry (QLineEdit) – поле ввода страны происхождения.
6. txtName (QLineEdit) – поле ввода названия производителя.

DealershipDialog (QDialog):

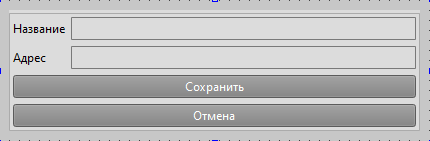


Рисунок 13 – DealershipDialog (QDialog)

1. btnCancel (QPushButton) – позволяет отменить операцию добавления нового транспортного средства.
2. btnOk (QPushButton) – вносит введенные изменения в БД.
3. label (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Название.
4. label\_2 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Адрес.
5. txtAddress (QLineEdit) – поле ввода адреса дилерского центра.
6. txtName (QLineEdit) – поле ввода названия дилерского центра.

CustomerDialog (QDialog):

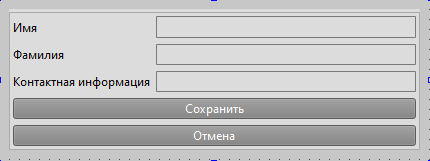


Рисунок 14 – CustomerDialog (QDialog)

1. btnCancel (QPushButton) – позволяет отменить операцию добавления нового транспортного средства.
2. btnOk (QPushButton) – вносит введенные изменения в БД.
3. label (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Имя.
4. label\_2 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Фамилия.
5. label\_3 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Контактная информация.
6. txtContacts (QLineEdit) – поле ввода контактной информации
7. txtFName (QLineEdit) – поле ввода имени.
8. txtLName (QLineEdit) – поле ввода фамилии.

SaleDialog (QDialog):

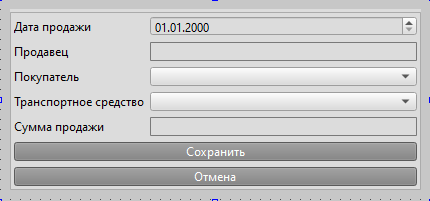


Рисунок 15 – SaleDialog (QDialog)

1. btnCancel (QPushButton) – позволяет отменить операцию добавления нового транспортного средства.
2. btnOk (QPushButton) – вносит введенные изменения в БД.
3. cbCustomer (QComboBox) – позволяет выбрать из базы данных нужного покупателя.
4. cbVehicle (QComboBox) – позволяет выбрать из базы данных нужное транспортное средство.
5. dateEdit (QDateEdit) – поле ввода даты продажи.
6. label (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Дата продажи.
7. label\_2 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Продавец.
8. label\_3 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Покупатель.
9. label\_4 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Транспортное средство.
10. label\_5 (QLabel) – информация о вводимой информации (то, что нужно вводить пользователю в поле ввода): Сумма продажи.
11. txtAmount (QLineEdit) – поле ввода продавца.
12. txtDealer (QLineEdit) – поле ввода суммы продажи.

**4.2. Реализация методов классов**

На данном этапе я внедрял методы для классов, которые были предварительно спроектированы на этапе создания классов. Разберем более подробно каждый из этих классов:

**4.2.1.** VehicleDialog:

Данный класс позволяет удобно добавлять новые данные о транспорте.

Класс **VehicleDialog** является подклассом **QDialog**. Класс содержит конструктор по умолчанию, деструктор и приватный указатель **ui** на объект класса **Ui::VehicleDialog**, который содержит описание пользовательского интерфейса формы добавления транспортного средства.

Также для него определены два приватных слота (**onBtnOkClicked** и **onBtnCancelClicked**), которые связаны с сигналами нажатия соответствующих кнопок на форме.

В конструкторе происходит инициализация объекта класса. Вызывается конструктор родительского класса **QDialog**. Также создается объект **ui** типа **Ui::VehicleDialog**, который инициализируется с использованием метода **setupUi(this)**. Деструктор освобождает выделенную память для объекта **ui**.

**4.2.2.** CustomerDialog:

Данный класс позволяет добавить новые данные о покупателе.

// Класс формы добавления покупателя

class CustomerDialog : public QDialog

{

Q\_OBJECT

// Конструктор

explicit CustomerDialog(QWidget \*parent = nullptr);

// Деструктор

~CustomerDialog();

В конструкторе происходит инициализация объекта класса.

Вызывается конструктор родительского класса **QDialog**. Также создается объект **ui** типа **Ui::CustomerDialog**, который инициализируется с использованием метода **setupUi(this)**. Этот метод применяет описание интерфейса к объекту **ui**.

Далее устанавливаются соединения (connect) между сигналами и слотами для кнопок "ОК" и "Отмена".

Деструктор освобождает выделенную память для объекта **ui**.

**4.2.3.** DbManager:

Класс **DbManager** представляет собой менеджер базы данных для взаимодействия с различными сущностями, такими как транспортные средства, производители, покупатели, дилерские центры и продажи. Для класса используется паттерн Singleton.

Методы и переменные данного класса:

* **QSqlDatabase db**: Объект для работы с базой данных.
* **connectToDB()**: Метод для подключения к базе данных.
* **getVehicles()**, **insertVehicle(Vehicle\* v)**: Методы для получения и добавления транспортных средств.
* Аналогичные методы для других сущностей (производителей, покупателей, дилерских центров, продаж).
* **openDataBase()**, **restoreDataBase()**, **closeDataBase()**: Методы для открытия, восстановления и закрытия базы данных.
* **createVehiclesTable()**, ...: Методы для создания таблиц в базе данных для различных сущностей.

**4.2.4.** DealershipDialog:

Класс **DealershipDialog** представляет собой диалоговое окно для добавления информации о дилерском центре, обрабатывает введенные данные.

// Класс формы добавления дилерского центра

class DealershipDialog : public QDialog

{

Q\_OBJECT

// Конструктор

explicit DealershipDialog(QWidget \*parent = nullptr);

// Деструктор

~DealershipDialog();

В конструкторе происходит инициализация объекта класса. Вызывается конструктор родительского класса **QDialog**. Также создается объект **ui** типа **Ui::DealershipDialog**, который инициализируется с использованием метода **setupUi(this)**.

Далее устанавливаются соединения (connect) между сигналами и слотами для кнопок "ОК" и "Отмена".

Деструктор освобождает выделенную память для объекта **ui**

**4.2.5** ManufacturerDialog:

Класс **ManufacturerDialog** представляет собой диалоговое окно для добавления информации о производителе, обрабатывает введенные данные.

// Класс формы добавления производителя

class ManufacturerDialog : public QDialog

{

Q\_OBJECT

// Конструктор

explicit ManufacturerDialog(QWidget \*parent = nullptr);

// Деструктор

~ManufacturerDialog();

В конструкторе происходит инициализация объекта класса. Вызывается конструктор родительского класса **QDialog**. Создается объект **ui** типа **Ui::ManufacturerDialog**, который инициализируется с использованием метода **setupUi(this)**.

Также устанавливаются соединения (connect) между сигналами и слотами для кнопок "ОК" и "Отмена".

Для каждого из классов применяются методы заполнения БД. В слотах кнопки OK прописаны методы заполнения базы данных в соответствии с классом.

**4.3. Разработка тестового приложения.**

После реализации основных классов, необходимо провести тестирование для убеждения в корректности их работы. В данном разделе представлено текстовое описание тестового приложения, созданного с целью проверки функциональности каждого класса.

VehicleDialog:

Тест с заполнением данных:

1. Передача параметров: марка, модель, год выпуска, стоимость, описание, статус.
2. Проверка корректности данных
3. Добавление в БД

Тест с пустым запросом:

1. Передача пустых параметров
2. Проверка корректности данных (ошибка)

Тест с неполным заполнением:

1. Передача данных
2. Проверка корректности (ошибка, не все данные заполнены)

ManufacturerDialog:

Тест с заполнением данных:

1. Передача данных: название, страна происхождения
2. Проверка корректности данных
3. Добавление в БД

Тест с пустым заполнением:

1. Передача пустых данных
2. Проверка корректности (ошибка)

Тест с неполным заполнением:

1. Передача данных
2. Проверка корректности (ошибка, не все данные заполнены)

DealershipDialog:

Тест с заполнением данных:

1. Передача данных: название, адрес
2. Проверка корректности данных
3. Добавление в БД

Тест с пустым заполнением:

1. Передача пустых данных
2. Проверка корректности (ошибка)

Тест с неполным заполнением:

1. Передача неполных данных
2. Проверка корректности (ошибка, не все данные заполнены)

CustomerDialog:

Тест с заполнением данных:

1. Передача данных: Имя, Фамилия, Контактная информация
2. Проверка корректности данных
3. Добавление в БД

Тест с пустым заполнением:

1. Передача пустых данных
2. Проверка корректности данных (ошибка)

Тест с неполным заполнением:

1. Передача неполных данных
2. Проверка корректности (ошибка, не все данные заполнены)

Тест с некорректным заполнением фамилии и имени

1. Ввод данных: фамилия, имя = цифровой ряд
2. Проверка корректности (вывод ошибки)

SaleDialog:

Тест с заполнением данных:

1. Передача данных: дата продажи, продавец, покупатель, т.с, сумма продажи
2. Проверка корректности данных
3. Добавление в БД

Тест с пустым заполнением:

1. Передача пустых данных
2. Проверка корректности данных (ошибка)

Тест с неполным заполнением:

1. Передача неполных данных
2. Проверка корректности (ошибка, не все данные заполнены)

Тест с некорректной суммой:

1. Ввод всех данных корректно, кроме “сумма продажи”
2. Проверка корректности (ошибка)

# 5. Тестирование

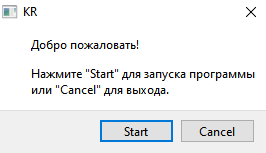


Рисунок 16 – запуск программы

Пользователя встречает окно приветствия, где предоставляется выбор: запустить систему либо выйти. При нажатии на кнопку “Start”, производится переход к основному функционалу программы.

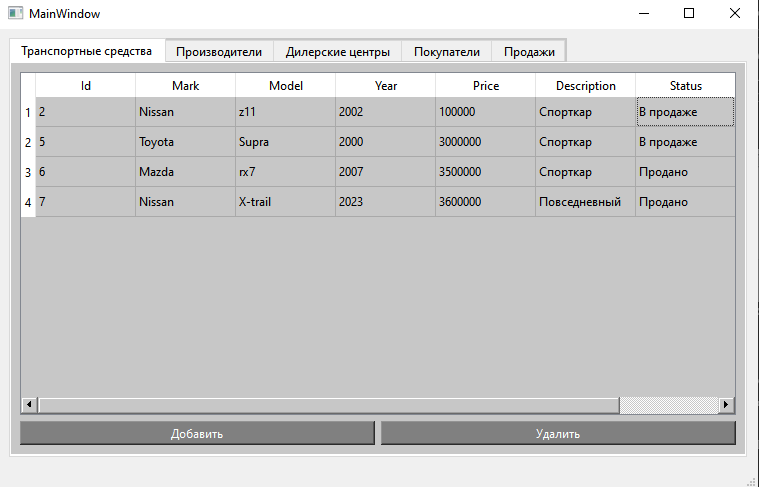


Рисунок 17 – меню управления программой (Транспортные средства)

В данном разделе программы пользователь вносит новые данные о транспорте, после чего информация о них поступает в базу данных и сохраняется. Чтобы добавить новый транспорт требуется нажать кнопку “Добавить”, чтобы удалить “Удалить”.

При нажатии на кнопку “Добавить” появится диалоговое окно (рисунок 18):

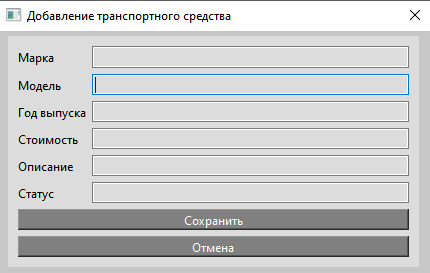


Рисунок 18 – добавление транспортного средства

Пользователь должен будет ввести данные об автомобиле и нажать кнопку сохранить. После этого данные будут внесены в БД.

Присутствует проверка на ввод:

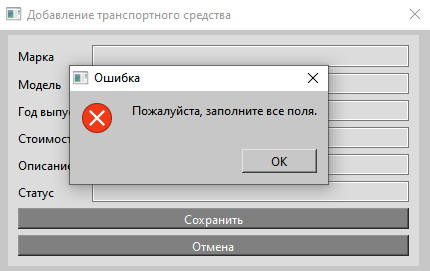


Рисунок 19 – ошибка (не все данные введены)

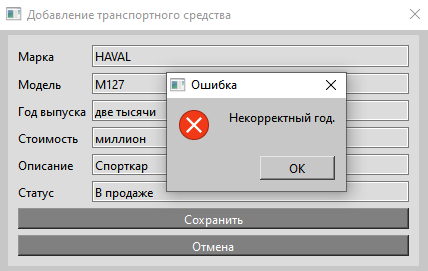


Рисунок 20 – ошибка (некорректный год)

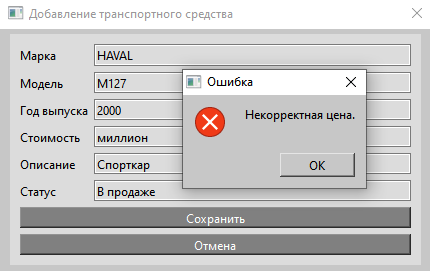


Рисунок 21 – ошибка (некорректная цена)

В случае успешного ввода данных:

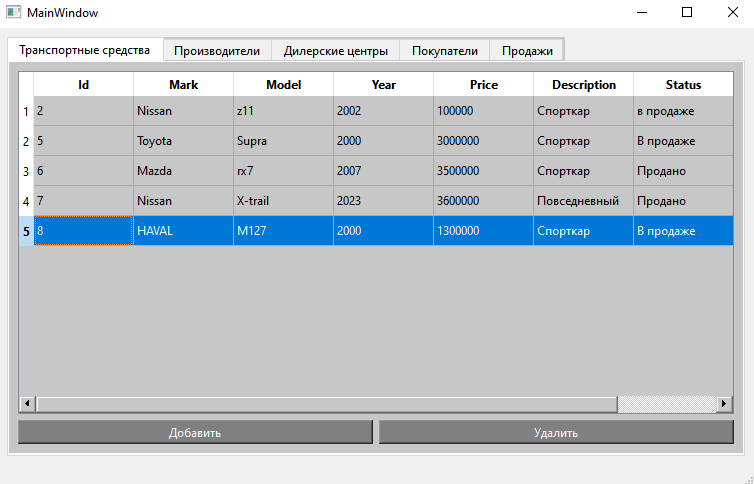


Рисунок 22 – успешное добавление транспорта

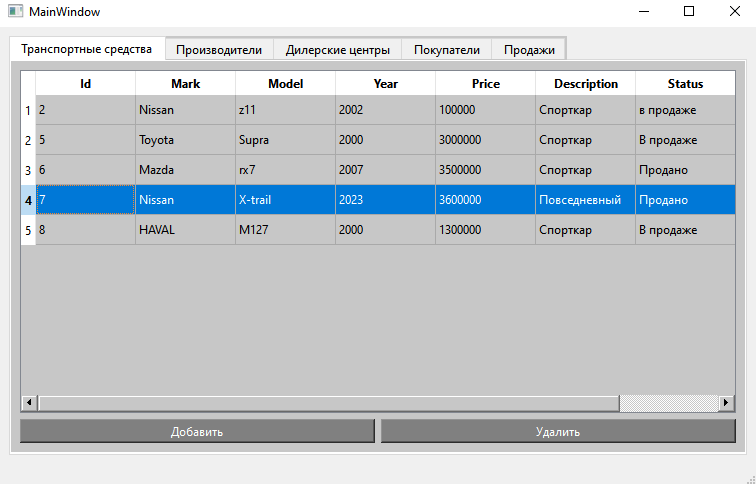


Рисунок 23 – удаление транспорта

Чтобы удалить транспорт, требуется выбрать нужный транспорт и нажать кнопку “Удалить”.

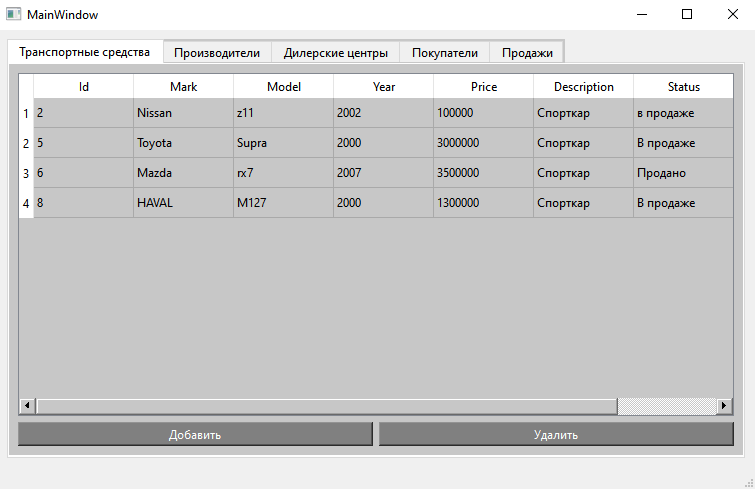


Рисунок 24 – успешное удаление

Транспорт успешно удален.

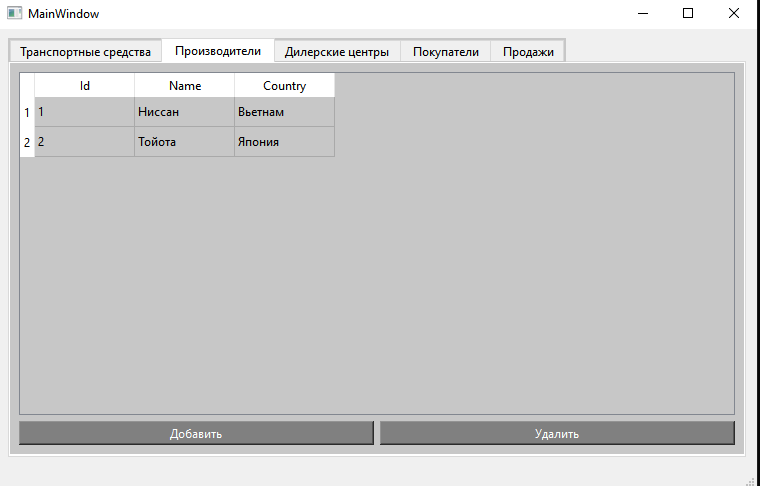


Рисунок 25 – производители

Во вкладке “Производители”, пользователь вносит данные о производителях автомобилей. Для этого требуется нажать кнопку “Добавить”, после чего откроется диалоговое окно:

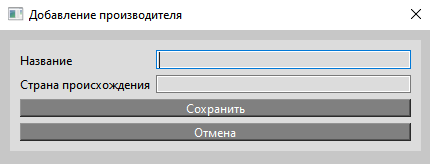


Рисунок 26 – добавление производителя

Здесь пользователь вводит название производителя и страну происхождения. Присутствуют проверки на ввод:

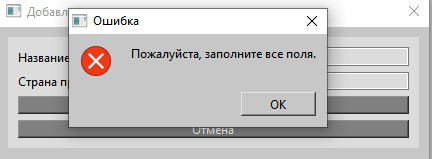


Рисунок 27 – ошибка (не все поля заполнены)

При успешно заполнении производитель будет успешно добавлен (рисунок 28):

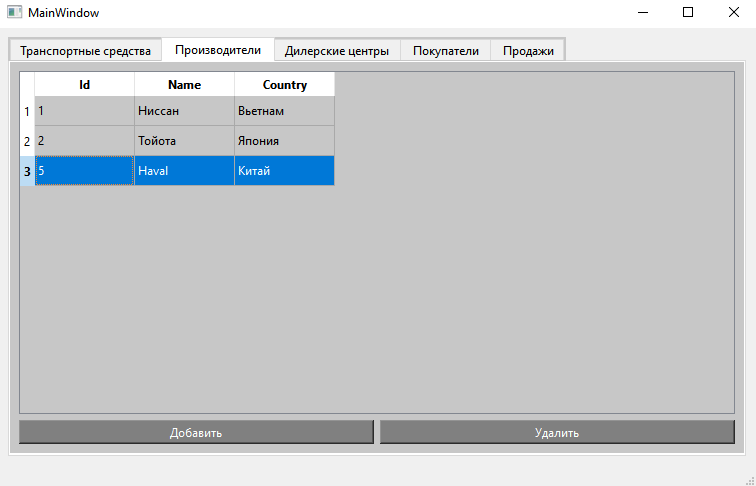


Рисунок 28 – успешное добавление производителя

Чтобы удалить производителя, требуется нажать кнопку “Удалить”, после чего выбранный производителей будет удален из БД (рисунок 29):

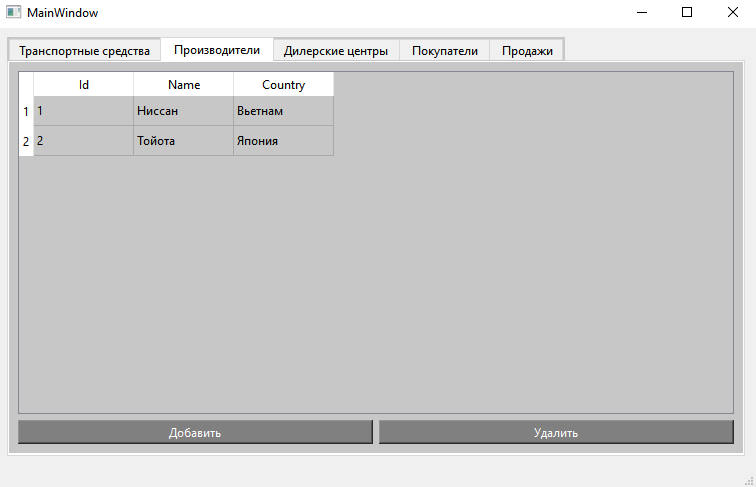


Рисунок 29 – успешное удаление производителя

Аналогично производится добавление дилерского центра (рисунок 30):

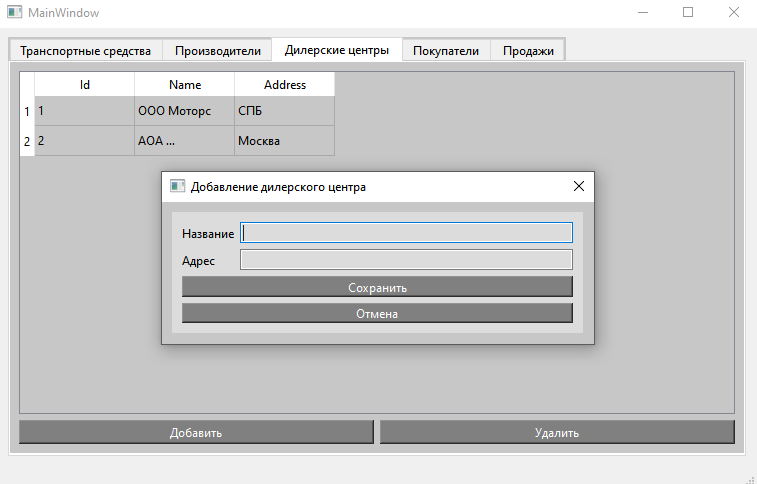


Рисунок 30 – добавление дилерского центра

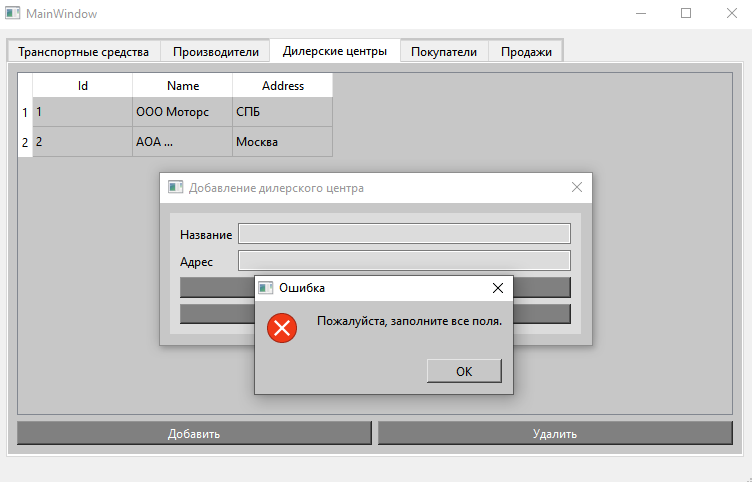


Рисунок 31 – проверка на корректность ввода

Пользователю необходимо ввести все данные для успешного добавления дилерского центра. В противном случае будет выведена ошибка.

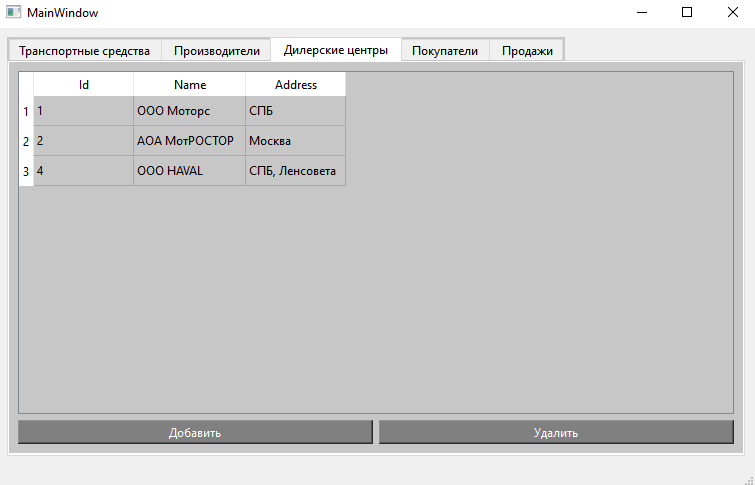


Рисунок 32 – успешное добавление дилерского центра

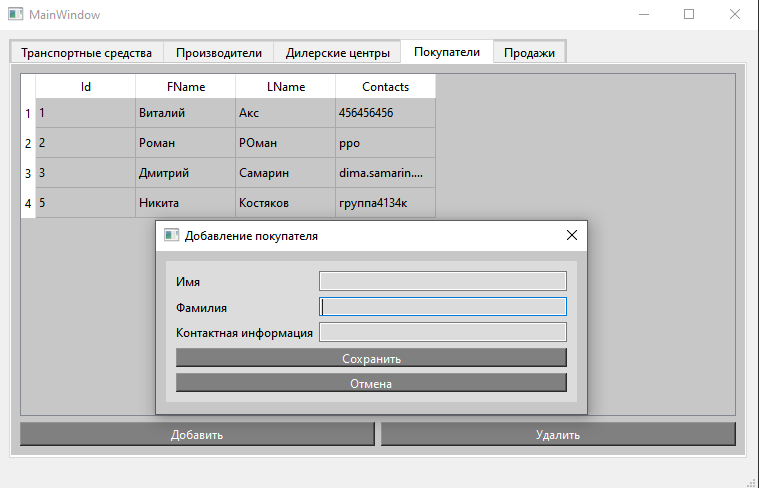


Рисунок 33 – вкладка покупатели, добавление

Вкладка покупатели позволяет добавить покупателя в базу данных. При добавлении производятся проверки на ввод:

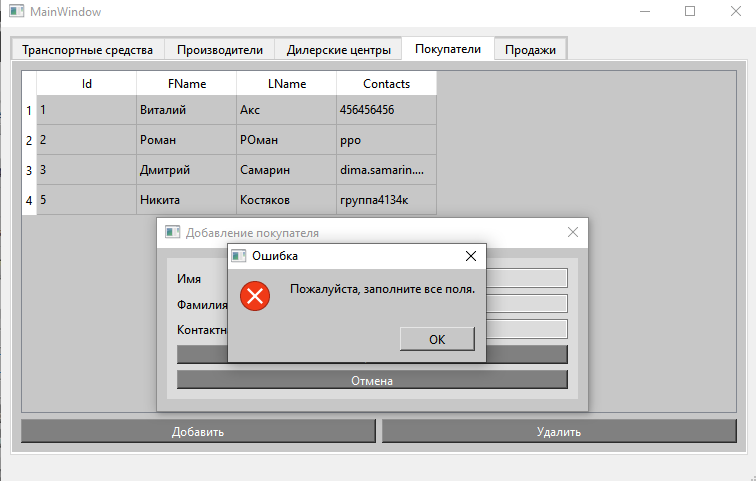


Рисунок 34 – ошибка (не все поля заполнены)

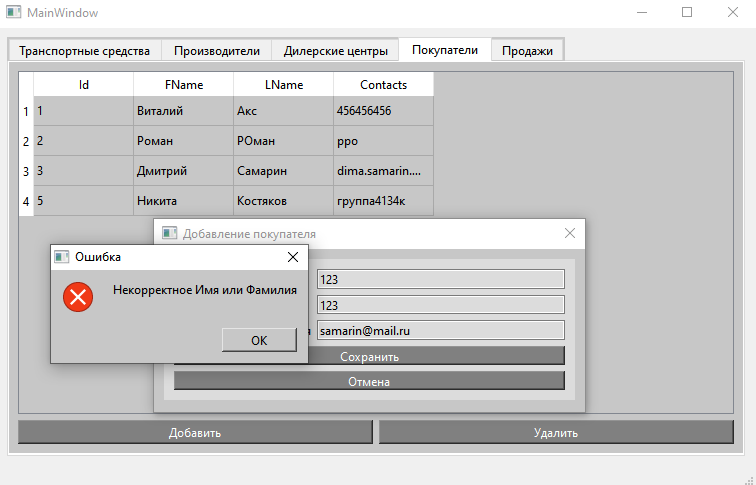


Рисунок 35 – ошибка (неправильный ввод фамилии либо имени)

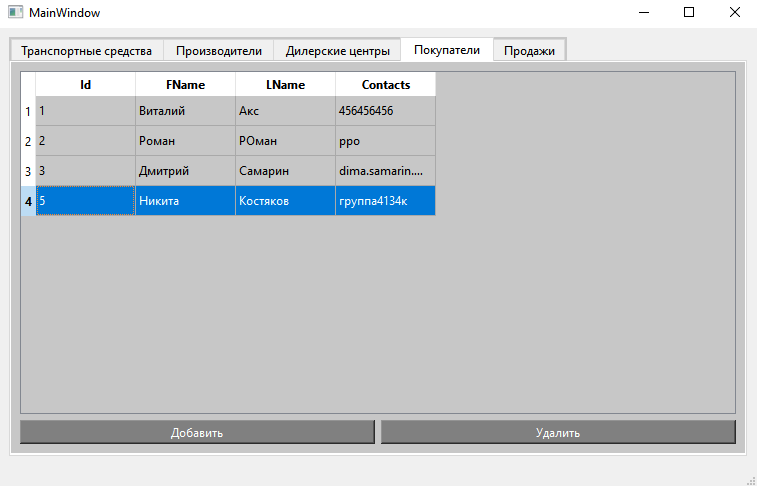


Рисунок 36 – удаление покупателя

Чтобы удалить покупателя, требуется выбрать его в списке покупателей и нажать кнопку “Удалить”.

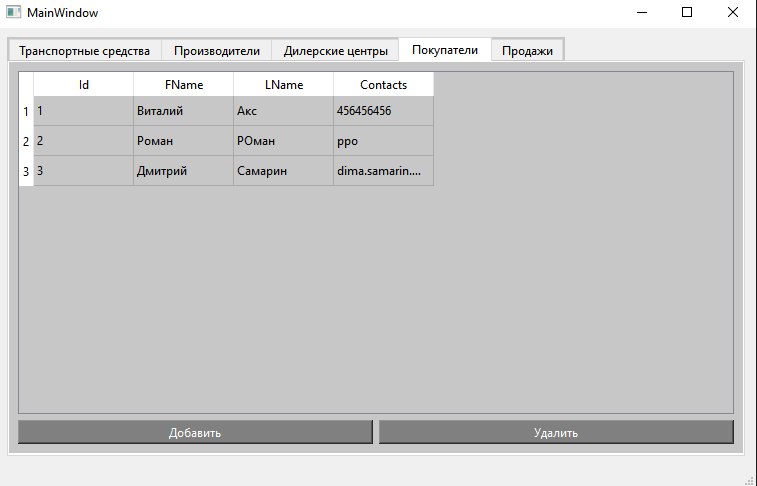


Рисунок 37 – успешное удаление

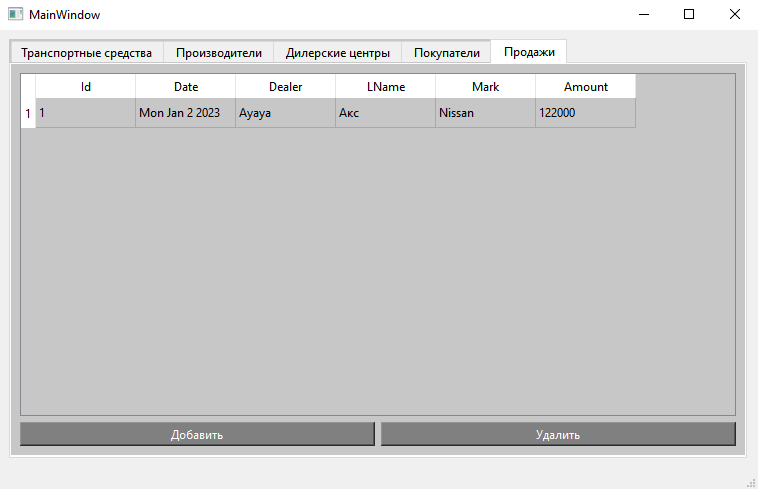


Рисунок 38 – вкладка продажи

Здесь производится внесение и отслеживание продаж автомобилей. Чтобы добавить новую продажу, требуется нажать кнопку “Добавить”:

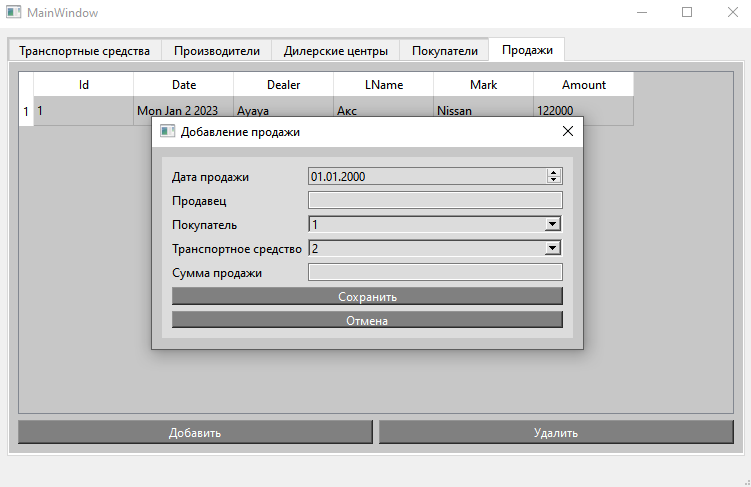


Рисунок 39 – добавление продажи

Чтобы внести продажу в БД, требуется ввести необходимые данные. При вводе производится проверка:

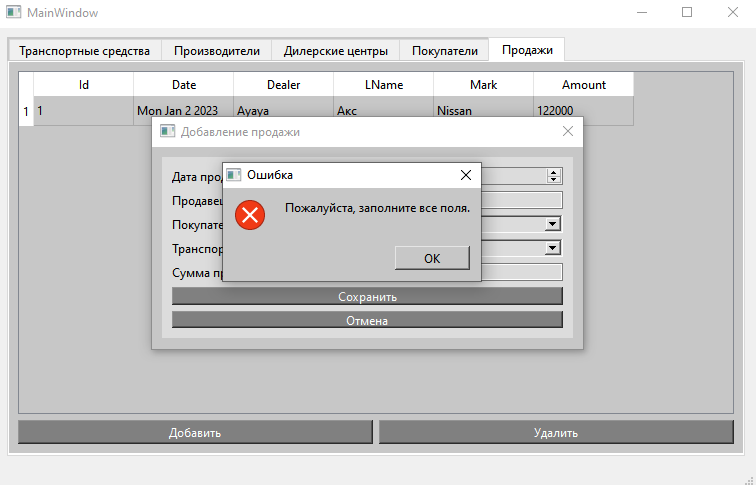


Рисунок 40 – ошибка (не все поля заполнены)

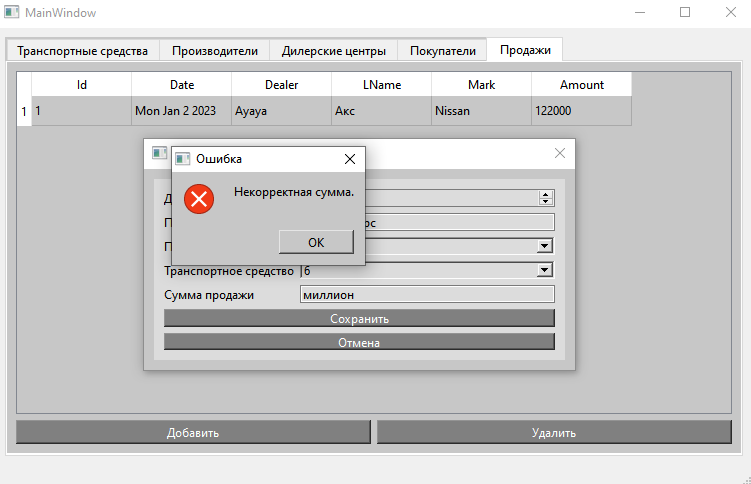


Рисунок 41 – ошибка (некорректная сумма)

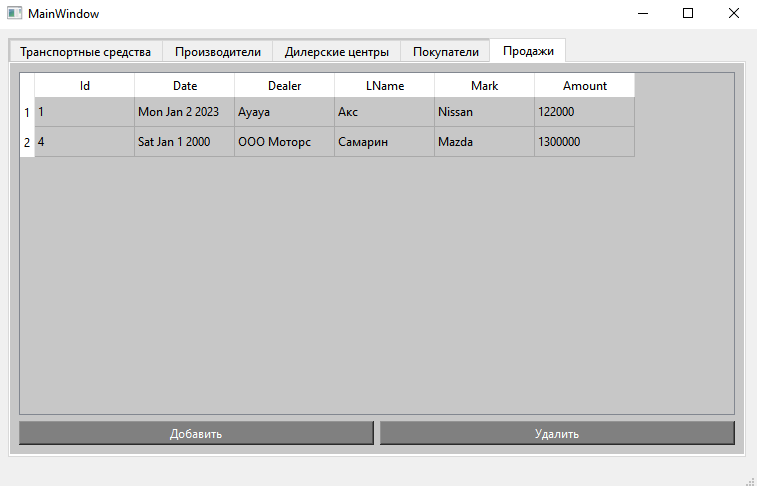


Рисунок 42 – успешное добавление продажи

Чтобы удалить из базы данных информацию о продаже требуется выбрать нужный вариант и нажать кнопку удалить.

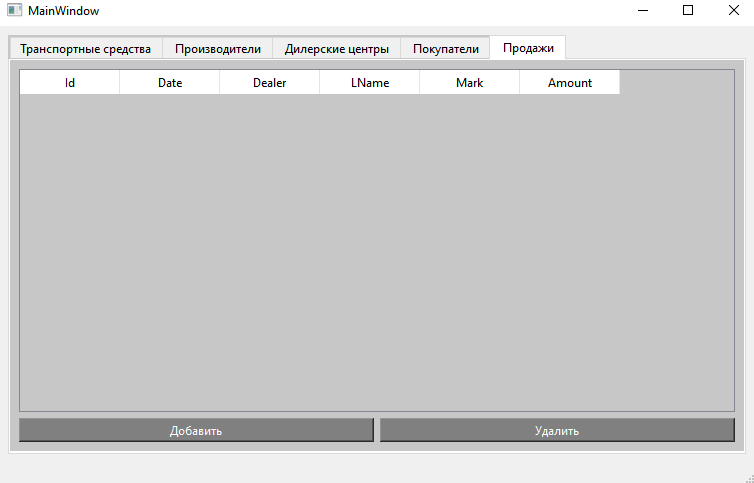


Рисунок 43 – успешное удаление

# 6. Приложение с полным кодом программ

customerdialog.cpp

#include "customerdialog.h"

#include "ui\_customerdialog.h"

#include "dbmanager.h"

#include "mainwindow.h"

// конструктор

CustomerDialog::CustomerDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::CustomerDialog) {

// создание UI

ui->setupUi(this);

// подключение сигналов к слотам

connect(ui->btnOk, SIGNAL(clicked()), this, SLOT(onBtnOkClicked()));

connect(ui->btnCancel, SIGNAL(clicked()), this, SLOT(onBtnCancelClicked()));

}

// деструктор

CustomerDialog::~CustomerDialog()

{

delete ui;

}

// слот кнопки ОК

void CustomerDialog::onBtnOkClicked(){

// Получаем значения из полей ввода

QString firstName = ui->txtFName->text().trimmed();

QString lastName = ui->txtLName->text().trimmed();

QString contacts = ui->txtContacts->text().trimmed();

// Проверка наличия данных в полях

if (firstName.isEmpty() || lastName.isEmpty() || contacts.isEmpty()) {

// Выводим сообщение об ошибке

QMessageBox::critical(this, "Ошибка", "Пожалуйста, заполните все поля.");

return;

}

// Проверка отсутствия цифр в полях "Имя" и "Фамилия"

QRegularExpression regex("\\d");

if (regex.match(firstName).hasMatch() || regex.match(lastName).hasMatch()) {

// Выводим сообщение об ошибке

QMessageBox::critical(this, "Ошибка", "Некорректное Имя или Фамилия");

return;

}

// Создаем объект Customer

Customer c;

c.first\_name = firstName;

c.last\_name = lastName;

c.contacts = contacts;

// Выполняем операции добавления в базу данных и обновления таблицы

DbManager::instance().insertCustomer(&c);

((MainWindow\*)parent())->updateTable();

// Закрываем диалоговое окно

this->close();

}

void CustomerDialog::onBtnCancelClicked(){

// Закрываем диалоговое окно

this->close();

}

Customerdialog.h

#ifndef CUSTOMERDIALOG\_H

#define CUSTOMERDIALOG\_H

#include <QDialog>

namespace Ui {

class CustomerDialog;

}

// класс формы добавления покупателя

class CustomerDialog : public QDialog

{

Q\_OBJECT

public:

// конструктор

explicit CustomerDialog(QWidget \*parent = nullptr);

// деструктор

~CustomerDialog();

private:

// указатель на ui

Ui::CustomerDialog \*ui;

private slots:

// слот кнопки ОК

void onBtnOkClicked();

// слот кнопки отмена

void onBtnCancelClicked();

};

#endif // CUSTOMERDIALOG\_H

Dbmanager.cpp

#include "dbmanager.h"

#include "dbnames.h"

DbManager::DbManager()

: Singleton<DbManager>(\*this) {

}

void DbManager::connectToDB() {

if(!QFile(DATABASE\_NAME).exists()){

this->restoreDataBase();

} else {

this->openDataBase();

}

}

bool DbManager::restoreDataBase() {

if(this->openDataBase()){

if(!this->createVehiclesTable() ||

!this->createCustomersTable() ||

!this->createManufacturersTable() ||

!this->createDealerShipsTable() ||

!this->createSalesTable()){

return false;

} else {

return true;

}

} else {

qDebug() << "Не удалось восстановить базу данных";

return false;

}

return false;

}

bool DbManager::openDataBase() {

db = QSqlDatabase::addDatabase("QSQLITE");

db.setDatabaseName(DATABASE\_NAME);

if(db.open()){

return true;

} else {

return false;

}

}

void DbManager::closeDataBase() {

db.close();

}

bool DbManager::createVehiclesTable(){

QSqlQuery query;

if(!query.exec( "CREATE TABLE " VEHICLES\_TABLE " ("

VEHICLE\_ID " INTEGER PRIMARY KEY AUTOINCREMENT, "

VEHICLE\_MARK " VARCHAR(255) NOT NULL,"

VEHICLE\_MODEL " VARCHAR(255) NOT NULL,"

VEHICLE\_YEAR " INTEGER NOT NULL,"

VEHICLE\_PRICE " INTEGER NOT NULL,"

VEHICLE\_DESCRIPTION " VARCHAR(255) NOT NULL,"

VEHICLE\_STATUS " VARCHAR(255) NOT NULL"

" )"

)){

qDebug() << "DataBase: error of create " << VEHICLES\_TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

bool DbManager::createManufacturersTable(){

QSqlQuery query;

if(!query.exec( "CREATE TABLE " MANUFACTURERS\_TABLE " ("

MANUFACTURER\_ID " INTEGER PRIMARY KEY AUTOINCREMENT, "

MANUFACTURER\_NAME " VARCHAR(255) NOT NULL,"

MANUFACTURER\_COUNTRY " VARCHAR(255) NOT NULL"

" )"

)){

qDebug() << "DataBase: error of create " << MANUFACTURERS\_TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

bool DbManager::createDealerShipsTable(){

QSqlQuery query;

if(!query.exec( "CREATE TABLE " DEALERSHIPS\_TABLE " ("

DEALERSHIP\_ID " INTEGER PRIMARY KEY AUTOINCREMENT, "

DEALERSHIP\_NAME " VARCHAR(255) NOT NULL,"

DEALERSHIP\_ADDRESS " VARCHAR(255) NOT NULL"

" )"

)){

qDebug() << "DataBase: error of create " << DEALERSHIPS\_TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

bool DbManager::createCustomersTable() {

QSqlQuery query;

if(!query.exec( "CREATE TABLE " CUSTOMERS\_TABLE " ("

CUSTOMER\_ID " INTEGER PRIMARY KEY AUTOINCREMENT, "

CUSTOMER\_FNAME " VARCHAR(255) NOT NULL,"

CUSTOMER\_LNAME " VARCHAR(255) NOT NULL,"

CUSTOMER\_CONTACTS " VARCHAR(255) NOT NULL"

" )"

)){

qDebug() << "DataBase: error of create " << CUSTOMERS\_TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

bool DbManager::createSalesTable(){

QSqlQuery query;

if(!query.exec( "CREATE TABLE " SALES\_TABLE " ("

SALE\_ID " INTEGER PRIMARY KEY AUTOINCREMENT, "

SALE\_DATE " TEXT NOT NULL,"

SALE\_DEALER " VARCHAR(255) NOT NULL,"

SALE\_CUSTOMER " INTEGER NOT NULL,"

SALE\_VEHICLE " INTEGER NOT NULL,"

SALE\_AMOUNT " VARCHAR(255) NOT NULL,"

"FOREIGN KEY (" SALE\_CUSTOMER ") REFERENCES " CUSTOMERS\_TABLE " (" CUSTOMER\_ID "),"

"FOREIGN KEY (" SALE\_AMOUNT ") REFERENCES " VEHICLES\_TABLE " (" VEHICLE\_ID ")"

" );"

)){

qDebug() << "DataBase: error of create " << SALES\_TABLE;

qDebug() << query.lastError().text();

return false;

} else {

return true;

}

return false;

}

QVector<Vehicle> DbManager::getVehicles() {

QVector<Vehicle> result;

QSqlQuery query;

if(!query.exec("SELECT \* FROM " VEHICLES\_TABLE)){

qDebug() << "DataBase: error of select " << VEHICLES\_TABLE;

return result;

}

QSqlRecord rec = query.record();

while (query.next()) {

Vehicle v;

v.id = query.value(rec.indexOf(VEHICLE\_ID)).toInt();

v.mark = query.value(rec.indexOf(VEHICLE\_MARK)).toString();

v.model = query.value(rec.indexOf(VEHICLE\_MODEL)).toString();

v.year = query.value(rec.indexOf(VEHICLE\_YEAR)).toInt();

v.price = query.value(rec.indexOf(VEHICLE\_PRICE)).toInt();

v.description = query.value(rec.indexOf(VEHICLE\_DESCRIPTION)).toString();

v.status = query.value(rec.indexOf(VEHICLE\_STATUS)).toString();

result.append(v);

}

return result;

}

bool DbManager::insertVehicle(Vehicle\* v) {

QString query\_string = "INSERT INTO " VEHICLES\_TABLE "("

VEHICLE\_MARK ", "

VEHICLE\_MODEL ", "

VEHICLE\_YEAR ", "

VEHICLE\_PRICE ", "

VEHICLE\_DESCRIPTION", "

VEHICLE\_STATUS ") "

"VALUES ('%1', '%2', %3, %4, '%5', '%6');";

query\_string = query\_string.arg(v->mark)

.arg(v->model)

.arg(v->year)

.arg(v->price)

.arg(v->description)

.arg(v->status);

QSqlQuery query;

if(!query.exec(query\_string)){

qDebug() << "DataBase: error of insert " << VEHICLES\_TABLE;

return false;

}

return true;

}

QVector<Manufacturer> DbManager::getManufacturers() {

QVector<Manufacturer> result;

QSqlQuery query;

if(!query.exec("SELECT \* FROM " MANUFACTURERS\_TABLE)){

qDebug() << "DataBase: error of select " << MANUFACTURERS\_TABLE;

return result;

}

QSqlRecord rec = query.record();

while (query.next()) {

Manufacturer m;

m.id = query.value(rec.indexOf(MANUFACTURER\_ID)).toInt();

m.name = query.value(rec.indexOf(MANUFACTURER\_NAME)).toString();

m.country = query.value(rec.indexOf(MANUFACTURER\_COUNTRY)).toString();

result.append(m);

}

return result;

}

bool DbManager::insertManufacturer(Manufacturer\* m) {

QString query\_string = "INSERT INTO " MANUFACTURERS\_TABLE "("

MANUFACTURER\_NAME ", "

MANUFACTURER\_COUNTRY ") "

"VALUES ('%1', '%2');";

query\_string = query\_string

.arg(m->name)

.arg(m->country);

QSqlQuery query;

if(!query.exec(query\_string)){

qDebug() << "DataBase: error of insert " << MANUFACTURERS\_TABLE;

return false;

}

return true;

}

QVector<Customer> DbManager::getCustomers() {

QVector<Customer> result;

QSqlQuery query;

if(!query.exec("SELECT \* FROM " CUSTOMERS\_TABLE)){

qDebug() << "DataBase: error of select " << CUSTOMERS\_TABLE;

return result;

}

QSqlRecord rec = query.record();

while (query.next()) {

Customer c;

c.id = query.value(rec.indexOf(CUSTOMER\_ID)).toInt();

c.first\_name = query.value(rec.indexOf(CUSTOMER\_FNAME)).toString();

c.last\_name = query.value(rec.indexOf(CUSTOMER\_LNAME)).toString();

c.contacts = query.value(rec.indexOf(CUSTOMER\_CONTACTS)).toString();

result.append(c);

}

return result;

}

bool DbManager::insertCustomer(Customer\* c) {

QString query\_string = "INSERT INTO " CUSTOMERS\_TABLE "("

CUSTOMER\_FNAME ", "

CUSTOMER\_LNAME ", "

CUSTOMER\_CONTACTS ") "

"VALUES ('%1', '%2', '%3');";

query\_string = query\_string

.arg(c->first\_name)

.arg(c->last\_name)

.arg(c->contacts);

QSqlQuery query;

if(!query.exec(query\_string)){

qDebug() << "DataBase: error of insert " << CUSTOMERS\_TABLE;

return false;

}

return true;

}

QVector<Dealership> DbManager::getDealerships() {

QVector<Dealership> result;

QSqlQuery query;

if(!query.exec("SELECT \* FROM " DEALERSHIPS\_TABLE)){

qDebug() << "DataBase: error of select " << DEALERSHIPS\_TABLE;

return result;

}

QSqlRecord rec = query.record();

while (query.next()) {

Dealership d;

d.id = query.value(rec.indexOf(DEALERSHIP\_ID)).toInt();

d.name = query.value(rec.indexOf(DEALERSHIP\_NAME)).toString();

d.address = query.value(rec.indexOf(DEALERSHIP\_ADDRESS)).toString();

result.append(d);

}

return result;

}

bool DbManager::insertDealership(Dealership\* d) {

QString query\_string = "INSERT INTO " DEALERSHIPS\_TABLE "("

DEALERSHIP\_NAME ", "

DEALERSHIP\_ADDRESS ") "

"VALUES ('%1', '%2');";

query\_string = query\_string

.arg(d->name)

.arg(d->address);

QSqlQuery query;

if(!query.exec(query\_string)){

qDebug() << "DataBase: error of insert " << DEALERSHIPS\_TABLE;

return false;

}

return true;

}

QSqlQuery DbManager::getSales() {

QString query\_string = "SELECT t1." SALE\_ID

", t1." SALE\_DATE

", t1." SALE\_DEALER

", t2." CUSTOMER\_LNAME

", t3." VEHICLE\_MARK

", t1." SALE\_AMOUNT

" FROM " SALES\_TABLE " t1 "

"JOIN " CUSTOMERS\_TABLE " t2 ON t1." SALE\_CUSTOMER "=t2." CUSTOMER\_ID " "

"JOIN " VEHICLES\_TABLE " t3 ON t1." SALE\_VEHICLE "=t3." VEHICLE\_ID ";";

QSqlQuery query;

query.exec(query\_string);

return query;

}

bool DbManager::insertSale(Sale\* s) {

QString query\_string = "INSERT INTO " SALES\_TABLE "("

SALE\_DATE ", "

SALE\_DEALER ", "

SALE\_CUSTOMER ", "

SALE\_VEHICLE ", "

SALE\_AMOUNT ") "

"VALUES ('%1', '%2', '%3', '%4', %5);";

query\_string = query\_string

.arg(s->date.toString())

.arg(s->dealer)

.arg(s->customer)

.arg(s->vehicle)

.arg(s->amount);

QSqlQuery query;

if(!query.exec(query\_string)){

qDebug() << "DataBase: error of insert " << SALES\_TABLE;

return false;

}

return true;

}

Dbmanager.h

#ifndef DBMANAGER\_H

#define DBMANAGER\_H

#include <QObject>

#include <QSql>

#include <QSqlQuery>

#include <QSqlError>

#include <QSqlDatabase>

#include <QFile>

#include <QDate>

#include <QDebug>

#include <QVector>

#include <QSqlRecord>

#include "model.h"

// шаблон для реализации паттерна синглтон

template <class T>

class Singleton {

public:

// конструктор

Singleton(T& rObject) {

Q\_ASSERT\_X(!s\_pInstance, "constructor", "Only one instance of this class is permitted.");

s\_pInstance = &rObject;

}

// деструктор

~Singleton() {

Q\_ASSERT\_X(s\_pInstance, "destructor", "The singleton instance is invalid.");

s\_pInstance = 0;

}

// получение ссылки на объект

static T& instance() {

if(!s\_pInstance) {

s\_pInstance = new T();

}

return (\*s\_pInstance);

}

private:

// указатель на объект

static T\* s\_pInstance;

// приватный конструктор

Singleton(const Singleton& Src);

// перегрузка оператора =

Singleton& operator=(const Singleton& Src);

};

template <class T> T\* Singleton<T>::s\_pInstance = 0;

class DbManager;

#define example DbManager::instance()

// класс для взаимодействия с БД

class DbManager : public Singleton<DbManager> {

public:

// конструктор

DbManager();

// метод подключения к БД

void connectToDB();

// получение всех транспортных средств

QVector<Vehicle> getVehicles();

// добавление тр. средства

bool insertVehicle(Vehicle\* v);

// получение всех производителей

QVector<Manufacturer> getManufacturers();

// добавление производителя

bool insertManufacturer(Manufacturer\* m);

// получение всех покупателей

QVector<Customer> getCustomers();

// добавление покупателя

bool insertCustomer(Customer\* c);

// получение всех дилерских центров

QVector<Dealership> getDealerships();

// добавление центра

bool insertDealership(Dealership\* d);

// получение всех продаж

QSqlQuery getSales();

// добавление продажи

bool insertSale(Sale\* s);

private:

// БД

QSqlDatabase db;

// открытие БД

bool openDataBase();

// восстановление БД

bool restoreDataBase();

// закрытие БД

void closeDataBase();

// методы создания таблиц в БД

bool createVehiclesTable();

bool createManufacturersTable();

bool createDealerShipsTable();

bool createCustomersTable();

bool createSalesTable();

};

#endif // DBMANAGER\_H

Dealershipdialog.cpp

#include "dealershipdialog.h"

#include "ui\_dealershipdialog.h"

#include "dbmanager.h"

#include "mainwindow.h"

DealershipDialog::DealershipDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::DealershipDialog) {

ui->setupUi(this);

connect(ui->btnOk, SIGNAL(clicked()), this, SLOT(onBtnOkClicked()));

connect(ui->btnCancel, SIGNAL(clicked()), this, SLOT(onBtnCancelClicked()));

}

DealershipDialog::~DealershipDialog() {

delete ui;

}

void DealershipDialog::onBtnOkClicked() {

// Получаем значения из полей ввода

QString name = ui->txtName->text().trimmed();

QString address = ui->txtAddress->text().trimmed();

// Проверка наличия данных в полях

if (name.isEmpty() || address.isEmpty()) {

// Выводим сообщение об ошибке

QMessageBox::critical(this, "Ошибка", "Пожалуйста, заполните все поля.");

return;

}

// Создаем объект Dealership

Dealership d;

d.name = name;

d.address = address;

// Выполняем операции добавления в базу данных и обновления таблицы

DbManager::instance().insertDealership(&d);

((MainWindow\*)parent())->updateTable();

// Закрываем диалоговое окно

this->close();

}

void DealershipDialog::onBtnCancelClicked() {

// Закрываем диалоговое окно

this->close();

}

Dealershipdialog.h

#ifndef DEALERSHIPDIALOG\_H

#define DEALERSHIPDIALOG\_H

#include <QDialog>

namespace Ui {

class DealershipDialog;

}

// класс формы добавления дилерского центра

class DealershipDialog : public QDialog

{

Q\_OBJECT

public:

explicit DealershipDialog(QWidget \*parent = nullptr);

~DealershipDialog();

private:

Ui::DealershipDialog \*ui;

private slots:

void onBtnOkClicked();

void onBtnCancelClicked();

};

#endif // DEALERSHIPDIALOG\_H

Dbnames.h

#ifndef DBNAMES\_H

#define DBNAMES\_H

// имя файла БД

#define DATABASE\_NAME "DataBase.db"

// имена таблиц БД

#define VEHICLES\_TABLE "Vehicles"

#define MANUFACTURERS\_TABLE "Manufacturers"

#define DEALERSHIPS\_TABLE "DealerShips"

#define CUSTOMERS\_TABLE "Customers"

#define SALES\_TABLE "Sales"

// имена полей в таблицах БД

#define VEHICLE\_ID "Id"

#define VEHICLE\_MARK "Mark"

#define VEHICLE\_MODEL "Model"

#define VEHICLE\_YEAR "Year"

#define VEHICLE\_PRICE "Price"

#define VEHICLE\_DESCRIPTION "Description"

#define VEHICLE\_STATUS "Status"

#define MANUFACTURER\_ID "Id"

#define MANUFACTURER\_NAME "Name"

#define MANUFACTURER\_COUNTRY "Country"

#define DEALERSHIP\_ID "Id"

#define DEALERSHIP\_NAME "Name"

#define DEALERSHIP\_ADDRESS "Address"

#define CUSTOMER\_ID "Id"

#define CUSTOMER\_FNAME "FName"

#define CUSTOMER\_LNAME "LName"

#define CUSTOMER\_CONTACTS "Contacts"

#define SALE\_ID "Id"

#define SALE\_DATE "Date"

#define SALE\_DEALER "Dealer"

#define SALE\_CUSTOMER "Customer"

#define SALE\_VEHICLE "Vehicle"

#define SALE\_AMOUNT "Amount"

#endif // DBNAMES\_H

Main.cpp

#include "mainwindow.h"

#include <QApplication>

#include <QMessageBox>

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

// Показываем окно приветствия

QMessageBox welcomeBox;

welcomeBox.setText("Добро пожаловать!");

welcomeBox.setInformativeText("Нажмите \"Start\" для запуска программы или \"Cancel\" для выхода.");

welcomeBox.setStandardButtons(QMessageBox::Ok | QMessageBox::Cancel);

welcomeBox.setDefaultButton(QMessageBox::Ok);

// Устанавливаем текст на кнопке Ok

auto okButton = welcomeBox.button(QMessageBox::Ok);

okButton->setText("Start");

if (welcomeBox.exec() == QMessageBox::Ok) {

// Если пользователь нажал "Start" в окне приветствия

MainWindow w;

w.show();

return a.exec();

} else {

// Если пользователь нажал "Отмена" в окне приветствия

return 0;

}

}

Mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QPushButton>

#include <QMessageBox>

#include <QtSql>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

// класс основного окна

class MainWindow : public QMainWindow {

Q\_OBJECT

public:

// конструктор

MainWindow(QWidget \*parent = nullptr);

//деструктор

~MainWindow();

// обновление таблиц

void updateTable();

private:

// указатель на UI

Ui::MainWindow \*ui;

// модель данных

QSqlTableModel \*model;

private slots:

// слоты для кнопок добавить и удалить

void onBtnAddVehicleClicked();

void onBtnRemoveVehicleClicked();

void onBtnAddManufacturerClicked();

void onBtnRemoveManufacturerClicked();

void onBtnAddCustomerClicked();

void onBtnRemoveCustomerClicked();

void onBtnAddDealershipClicked();

void onBtnRemoveDealershipClicked();

void onBtnAddSaleClicked();

void onBtnRemoveSaleClicked();

// слот смены вкладки

void tabSelected();

};

#endif // MAINWINDOW\_H

Mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

#include <iostream>

#include "dbnames.h"

#include "dbmanager.h"

#include "vehicledialog.h"

#include "manufacturerdialog.h"

#include "dealershipdialog.h"

#include "customerdialog.h"

#include "saledialog.h"

// конструктор

MainWindow::MainWindow(QWidget \*parent)

: QMainWindow(parent)

, ui(new Ui::MainWindow)

{

ui->setupUi(this);

// подключаемся к БД

DbManager::instance().connectToDB();

// создаем модель данных

model = new QSqlTableModel;

// подключение сигналов к слотам

connect(ui->tabWidget, SIGNAL(currentChanged(int)), this, SLOT(tabSelected()));

connect(ui->btnAddVehicle, SIGNAL(clicked()), this, SLOT(onBtnAddVehicleClicked()));

connect(ui->btnRemoveVehicle, SIGNAL(clicked()), this, SLOT(onBtnRemoveVehicleClicked()));

connect(ui->btnAddManufacturer, SIGNAL(clicked()), this, SLOT(onBtnAddManufacturerClicked()));

connect(ui->btnRemoveManufacturer, SIGNAL(clicked()), this, SLOT(onBtnRemoveManufacturerClicked()));

connect(ui->btnAddCustomer, SIGNAL(clicked()), this, SLOT(onBtnAddCustomerClicked()));

connect(ui->btnRemoveCustomer, SIGNAL(clicked()), this, SLOT(onBtnRemoveCustomerClicked()));

connect(ui->btnAddDealership, SIGNAL(clicked()), this, SLOT(onBtnAddDealershipClicked()));

connect(ui->btnRemoveDealership, SIGNAL(clicked()), this, SLOT(onBtnRemoveDealershipClicked()));

connect(ui->btnAddSale, SIGNAL(clicked()), this, SLOT(onBtnAddSaleClicked()));

connect(ui->btnRemoveSale, SIGNAL(clicked()), this, SLOT(onBtnRemoveSaleClicked()));

updateTable();

}

// деструктор

MainWindow::~MainWindow() {

delete model;

delete ui;

}

// слот смены вкладки

void MainWindow::tabSelected(){

updateTable();

}

// метод обновления таблиц

void MainWindow::updateTable(){

if(ui->tabWidget->currentIndex()==0){

model->setTable(VEHICLES\_TABLE);

model->select();

ui->tableVehicle->setModel(model);

}

if(ui->tabWidget->currentIndex()==1){

model->setTable(MANUFACTURERS\_TABLE);

model->select();

ui->tableManufacturers->setModel(model);

}

if(ui->tabWidget->currentIndex()==2){

model->setTable(DEALERSHIPS\_TABLE);

model->select();

ui->tableDealerships->setModel(model);

}

if(ui->tabWidget->currentIndex()==3){

model->setTable(CUSTOMERS\_TABLE);

model->select();

ui->tableCustomers->setModel(model);

}

if(ui->tabWidget->currentIndex()==4){

QSqlQuery q = DbManager::instance().getSales();

QSqlQueryModel \*model = new QSqlQueryModel();

model->setQuery(std::move(q));

ui->tableSales->setModel(model);

}

}

// слоты для кнопок добавить и удалить

void MainWindow::onBtnAddVehicleClicked(){

VehicleDialog\* dialog = new VehicleDialog(this);

dialog->show();

}

void MainWindow::onBtnRemoveVehicleClicked(){

int selected\_row\_index = ui->tableVehicle->currentIndex().row();

if(selected\_row\_index < 0)

return;

model->removeRows(selected\_row\_index, 1);

updateTable();

}

void MainWindow::onBtnAddManufacturerClicked(){

ManufacturerDialog\* dialog = new ManufacturerDialog(this);

dialog->show();

}

void MainWindow::onBtnRemoveManufacturerClicked(){

int selected\_row\_index = ui->tableManufacturers->currentIndex().row();

if(selected\_row\_index < 0)

return;

model->removeRows(selected\_row\_index, 1);

updateTable();

}

void MainWindow::onBtnAddCustomerClicked(){

CustomerDialog\* dialog = new CustomerDialog(this);

dialog->show();

}

void MainWindow::onBtnRemoveCustomerClicked(){

int selected\_row\_index = ui->tableCustomers->currentIndex().row();

if(selected\_row\_index < 0)

return;

model->removeRows(selected\_row\_index, 1);

updateTable();

}

void MainWindow::onBtnAddDealershipClicked(){

DealershipDialog\* dialog = new DealershipDialog(this);

dialog->show();

}

void MainWindow::onBtnRemoveDealershipClicked(){

int selected\_row\_index = ui->tableDealerships->currentIndex().row();

if(selected\_row\_index < 0)

return;

model->removeRows(selected\_row\_index, 1);

updateTable();

}

void MainWindow::onBtnAddSaleClicked(){

SaleDialog\* dialog = new SaleDialog(this);

dialog->show();

}

void MainWindow::onBtnRemoveSaleClicked(){

int selected\_row\_index = ui->tableSales->currentIndex().row();

if(selected\_row\_index < 0)

return;

model->removeRows(selected\_row\_index, 1);

updateTable();

}

Manufacturerdialog.cpp

#include "manufacturerdialog.h"

#include "ui\_manufacturerdialog.h"

#include "dbmanager.h"

#include "mainwindow.h"

// конструктор

ManufacturerDialog::ManufacturerDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::ManufacturerDialog) {

ui->setupUi(this);

connect(ui->btnOk, SIGNAL(clicked()), this, SLOT(onBtnOkClicked()));

connect(ui->btnCancel, SIGNAL(clicked()), this, SLOT(onBtnCancelClicked()));

}

// деструктор

ManufacturerDialog::~ManufacturerDialog() {

delete ui;

}

// слоты для кнопок ОК и отмена

void ManufacturerDialog::onBtnOkClicked() {

// Получаем значения из полей ввода

QString name = ui->txtName->text().trimmed();

QString country = ui->txtCountry->text().trimmed();

// Проверка наличия данных в полях

if (name.isEmpty() || country.isEmpty()) {

// Выводим сообщение об ошибке

QMessageBox::critical(this, "Ошибка", "Пожалуйста, заполните все поля.");

return;

}

// Создаем объект Manufacturer

Manufacturer m;

m.name = name;

m.country = country;

// Выполняем операции добавления в базу данных и обновления таблицы

DbManager::instance().insertManufacturer(&m);

((MainWindow\*)parent())->updateTable();

// Закрываем диалоговое окно

this->close();

}

void ManufacturerDialog::onBtnCancelClicked() {

// Закрываем диалоговое окно

this->close();

}

Manufacturerdialog.h

#ifndef MANUFACTURERDIALOG\_H

#define MANUFACTURERDIALOG\_H

#include <QDialog>

namespace Ui {

class ManufacturerDialog;

}

// класс формы добавления производителя

class ManufacturerDialog : public QDialog

{

Q\_OBJECT

public:

explicit ManufacturerDialog(QWidget \*parent = nullptr);

~ManufacturerDialog();

private:

Ui::ManufacturerDialog \*ui;

private slots:

void onBtnOkClicked();

void onBtnCancelClicked();

};

#endif // MANUFACTURERDIALOG\_H

Saledialog.cpp

#include "saledialog.h"

#include "ui\_saledialog.h"

#include <QSqlTableModel>

#include <QTableView>

#include <QtSql>

#include <QAbstractItemView>

#include <QMessageBox> // Добавьте этот заголовочный файл для использования QMessageBox

#include "dbmanager.h"

#include "mainwindow.h"

#include "dbnames.h"

// конструктор

SaleDialog::SaleDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::SaleDialog) {

ui->setupUi(this);

// создаем модель

QSqlQueryModel \*model1 = new QSqlQueryModel;

// передаем в модель текст sql запроса

model1->setQuery("SELECT " CUSTOMER\_ID ", " CUSTOMER\_LNAME " FROM " CUSTOMERS\_TABLE);

// устанавливаем столбцы

model1->setHeaderData(0,Qt::Horizontal, tr(CUSTOMER\_ID));

model1->setHeaderData(1, Qt::Horizontal, tr(CUSTOMER\_LNAME));

// создаем таблицу, которая будет отображаться в комбобоксе

QTableView \*view1 = new QTableView;

// настраиваем таблицу

view1->setSelectionBehavior(QAbstractItemView::SelectRows);

view1->setSelectionMode(QAbstractItemView::SingleSelection);

view1->showColumn(1);

// передаем модель и таблицу в комбобокс

ui->cbCustomer->setModel(model1);

ui->cbCustomer->setView(view1);

QSqlQueryModel \*model2 = new QSqlQueryModel;

model2->setQuery("SELECT " VEHICLE\_ID ", " VEHICLE\_MARK ", " VEHICLE\_MODEL " FROM " VEHICLES\_TABLE);

model2->setHeaderData(0,Qt::Horizontal, tr(VEHICLE\_ID));

model2->setHeaderData(1, Qt::Horizontal, tr(VEHICLE\_MARK));

model2->setHeaderData(2, Qt::Horizontal, tr(VEHICLE\_MODEL));

QTableView \*view2 = new QTableView;

view2->setSelectionBehavior(QAbstractItemView::SelectRows);

view2->setSelectionMode(QAbstractItemView::SingleSelection);

view2->setColumnHidden(0, true);

ui->cbVehicle->setModel(model2);

ui->cbVehicle->setView(view2);

connect(ui->btnOk, SIGNAL(clicked()), this, SLOT(onBtnOkClicked()));

connect(ui->btnCancel, SIGNAL(clicked()), this, SLOT(onBtnCancelClicked()));

}

SaleDialog::~SaleDialog()

{

delete ui;

}

void SaleDialog::onBtnOkClicked(){

// Проверка на ввод данных

if (ui->dateEdit->date().isNull() ||

ui->txtDealer->text().isEmpty() ||

ui->cbCustomer->currentText().isEmpty() ||

ui->cbVehicle->currentText().isEmpty() ||

ui->txtAmount->text().isEmpty()) {

QMessageBox::critical(this, "Ошибка", "Пожалуйста, заполните все поля.");

return;

}

Sale s;

s.date = ui->dateEdit->date();

s.dealer = ui->txtDealer->text();

// Проверка на ввод числовых данных

bool conversionOk = false;

s.customer = ui->cbCustomer->currentText().toInt(&conversionOk);

if (!conversionOk) {

QMessageBox::critical(this, "Ошибка", "Некорректный ID клиента.");

return;

}

s.vehicle = ui->cbVehicle->currentText().toInt(&conversionOk);

if (!conversionOk) {

QMessageBox::critical(this, "Ошибка", "Некорректный ID транспортного средства.");

return;

}

s.amount = ui->txtAmount->text().toInt(&conversionOk);

if (!conversionOk) {

QMessageBox::critical(this, "Ошибка", "Некорректная сумма.");

return;

}

DbManager::instance().insertSale(&s);

((MainWindow\*)parent())->updateTable();

this->close();

}

void SaleDialog::onBtnCancelClicked(){

this->close();

}

Model.h

#ifndef MODEL\_H

#define MODEL\_H

#include <QString>

#include <QDate>

// классы модели

class Vehicle {

public:

int id;

QString mark;

QString model;

int year;

int price;

QString description;

QString status;

};

class Manufacturer{

public:

int id;

QString name;

QString country;

};

class Dealership {

public:

int id;

QString name;

QString address;

};

class Customer {

public:

int id;

QString first\_name;

QString last\_name;

QString contacts;

};

class Sale {

public:

int id;

QDate date;

QString dealer;

int customer;

int vehicle;

int amount;

};

#endif // MODEL\_H

Vehicledialog.cpp

#include "vehicledialog.h"

#include "ui\_vehicledialog.h"

#include <QDebug>

#include <QMessageBox> // Добавьте этот заголовочный файл для использования QMessageBox

#include "dbmanager.h"

#include "mainwindow.h"

// конструктор

VehicleDialog::VehicleDialog(QWidget \*parent) :

QDialog(parent),

ui(new Ui::VehicleDialog) {

ui->setupUi(this);

connect(ui->btnOk, SIGNAL(clicked()), this, SLOT(onBtnOkClicked()));

connect(ui->btnCancel, SIGNAL(clicked()), this, SLOT(onBtnCancelClicked()));

}

// деструктор

VehicleDialog::~VehicleDialog()

{

delete ui;

}

// слоты для кнопок

void VehicleDialog::onBtnOkClicked(){

// Проверка на ввод данных

if (ui->txtMark->text().isEmpty() ||

ui->txtModel->text().isEmpty() ||

ui->txtYear->text().isEmpty() ||

ui->txtPrice->text().isEmpty() ||

ui->txtDescription->text().isEmpty() ||

ui->txtStatus->text().isEmpty()) {

QMessageBox::critical(this, "Ошибка", "Пожалуйста, заполните все поля.");

return;

}

Vehicle v;

v.mark = ui->txtMark->text();

v.model = ui->txtModel->text();

// Проверка на ввод числовых данных

bool conversionOk = false;

v.year = ui->txtYear->text().toInt(&conversionOk);

if (!conversionOk) {

QMessageBox::critical(this, "Ошибка", "Некорректный год.");

return;

}

v.price = ui->txtPrice->text().toInt(&conversionOk);

if (!conversionOk) {

QMessageBox::critical(this, "Ошибка", "Некорректная цена.");

return;

}

v.description = ui->txtDescription->text();

v.status = ui->txtStatus->text();

DbManager::instance().insertVehicle(&v);

((MainWindow\*)parent())->updateTable();

this->close();

}

void VehicleDialog::onBtnCancelClicked(){

this->close();

}

Saledialog.h

#ifndef SALEDIALOG\_H

#define SALEDIALOG\_H

#include <QDialog>

namespace Ui {

class SaleDialog;

}

// класс формы добавления продажи

class SaleDialog : public QDialog

{

Q\_OBJECT

public:

explicit SaleDialog(QWidget \*parent = nullptr);

~SaleDialog();

private:

Ui::SaleDialog \*ui;

private slots:

void onBtnOkClicked();

void onBtnCancelClicked();

};

#endif // SALEDIALOG\_H

Vehicledialog.h

#ifndef SALEDIALOG\_H

#define SALEDIALOG\_H

#include <QDialog>

namespace Ui {

class SaleDialog;

}

// класс формы добавления продажи

class SaleDialog : public QDialog

{

Q\_OBJECT

public:

explicit SaleDialog(QWidget \*parent = nullptr);

~SaleDialog();

private:

Ui::SaleDialog \*ui;

private slots:

void onBtnOkClicked();

void onBtnCancelClicked();

};

#endif // SALEDIALOG\_H

# 7. Заключение

Выполнив данный проект мной, была конструирована система согласно поставленной задаче. В процессе разработки были выявлены ключевые сущности предметной области и изучены их атрибуты и взаимосвязи. Была реализована функциональность позволяющая вести учет продаж транспортных средств, что в реалиях нашего мира является важным аспектом дилерских компаний.

# 8. Список использованных источников

1. <https://runebook.dev/ru/docs/qt/-index-> - документация QT русский вариант
2. <https://doc.qt.io/> - Официальная документация QT
3. Мэтт Вайсфельд «Объектно-ориентированное мышление»
4. <https://learn.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170> – Справочные материалы по С++ от Microsoft
5. <https://metanit.com/cpp/tutorial/> - Руководство по программированию на С++
6. <https://tproger.ru/translations/design-patterns-for-beginners> - Статья про паттерны проектирования