Министерство образования и науки Российской Федерации

**Муромский институт (филиал)**

федерального государственного бюджетного образовательного учреждения

высшего профессионального образования

**«Владимирский государственный университет**

**имени Александра Григорьевича и Николая Григорьевича Столетовых»**

(МИ (филиал) ВлГУ)

Факультет ИТР

Кафедра ФПМ

*КУРСОВАЯ РАБОТА*

по дисциплине СУБД

Тема ИС магазина продажи телефонов

Руководитель

(оценка)  (уч. степень, звание)

(подпись) (дата)

Члены комиссии Студент ПМИ-122

(группа)

Парнюгин Н.А.

(подпись) (дата)

(подпись) (дата) (подпись) (дата)

Муром 2024 г.

В работе представлена информационная система магазина продажи телефонов. Приложение разработано на платформе Windows. Цель работы - рассмотреть принципы разработки информационных систем и разработать программу на основе этих принципов.

Для разработки были выбраны язык программирования - Python и среда разработки – PyCharm Commumity Edition. Для создания базы данных использовалась СУБД SQLite.

В разработанном приложении доступны возможность добавления, удаления, изменения и поиска записей в базе данных.

The paper presents the information system of a phone sales store. The application is developed on the Windows platform. The purpose of the work is to consider the principles of information system development and develop a program based on these principles.

The programming language Python and the development environment PyCharm Commodity Edition were chosen for development. The SQLite database was used to create the database.

The developed application provides the ability to add, delete, modify, and search for records in the database.

Содержание.

[Введение 6](#_Toc440226278)

[1. Анализ технического задания 9](#_Toc440226279)

[2. Разработка моделей данных. 12](#_Toc440226280)

[2.1 Концептуальная модель 13](#_Toc440226281)

[2.2 Логическая модель 14](#_Toc440226282)

[2.3 Приведение логической модели к физической 15](#_Toc440226284)

[3. Разработка и реализация ИС. 16](#_Toc440226285)

[3.1 Разработка средств защиты данных и программ 16](#_Toc440226286)

[3.2. Разработка серверной части 18](#_Toc440226287)

3.3 Разработка SQL – запросов………………………………………………..20

3.4Разработка клиентской части…………………………………………..24

3.5 Руководство пользователю и программисту……………………………..29

[4. Тестирование ИС. 31](#_Toc440226288)

Заключение……………………………………………………………………….33

Список литературы………………………………………………………………34

Приложение 1…………………………………………………………………….35

Приложение 2…………………………………………………………………….37

Введение:

Данная курсовая работа представляет собой подробное исследование и разработку информационной системы для магазина, специализирующегося на продаже бывших в употреблении мобильных телефонов и новых аксессуаров к ним, а также предоставляющего услуги по обслуживанию мобильных устройств. Актуальность данной работы обусловлена растущим спросом на мобильные устройства, в том числе подержанные, и необходимостью эффективного управления магазинами, работающими в этой сфере. В условиях динамично развивающегося рынка автоматизация процессов учета и управления является ключевым фактором для обеспечения конкурентоспособности и прибыльности бизнеса.

Основной целью данной курсовой работы является создание полноценной информационной системы, позволяющей автоматизировать ключевые процессы магазина, включая учёт телефонов, аксессуаров, услуг, а также формирование необходимых отчётов. Работа выполнена в рамках изучения дисциплины «Системы управления базами данных» и демонстрирует практическое применение полученных знаний в области проектирования и разработки баз данных, а также разработки пользовательских приложений.

**1. Актуальность и постановка проблемы**

**1.1 Актуальность разработки информационной системы**

Современный рынок мобильных устройств характеризуется высоким уровнем конкуренции и быстрым темпом изменений. Магазины, торгующие телефонами и аксессуарами, вынуждены постоянно адаптироваться к новым условиям, чтобы оставаться конкурентоспособными. В частности, растет спрос на подержанные телефоны, что делает этот сегмент рынка все более значимым.

В таких условиях использование информационных систем для автоматизации процессов учёта и управления становится не просто желательным, а необходимым. Информационная система позволяет:

* **Повысить эффективность работы персонала:** автоматизация рутинных задач, таких как учет товаров и формирование отчетов, позволяет персоналу сосредоточиться на более важных задачах, например, на обслуживании клиентов.
* **Сократить время обработки данных:** компьютеризированные системы позволяют быстро и точно обрабатывать большие объемы данных, что сокращает время выполнения операций и ускоряет принятие решений.
* **Минимизировать количество ошибок:** ручной ввод данных чреват ошибками, которые могут привести к неправильному учету товаров и финансовым потерям. Автоматизированные системы минимизируют вероятность таких ошибок.
* **Предоставить руководству магазина возможность анализировать продажи и прибыль:** информационная система позволяет собирать данные о продажах и прибыли, которые можно использовать для анализа рыночных тенденций и принятия обоснованных управленческих решений.
* **Улучшить качество обслуживания клиентов:** быстрая и точная обработка заказов и предоставление актуальной информации о наличии товаров повышают удовлетворенность клиентов.

Таким образом, разработка информационной системы для магазина по продаже телефонов является актуальной и необходимой для обеспечения его эффективного и конкурентоспособного функционирования.

**1.2 Постановка проблемы и формулировка целей и задач**

Основной проблемой, которую призвана решить данная работа, является отсутствие автоматизированной системы управления магазином по продаже телефонов. В настоящее время многие магазины используют устаревшие методы учёта и управления, что приводит к низкой эффективности работы, ошибкам в учёте и управлении, а также к трудностям в анализе и принятии решений.

На основе выявленной проблемы, были сформулированы следующие цели и задачи работы:

**Цель:** разработка функциональной и удобной информационной системы для магазина по продаже телефонов, обеспечивающей автоматизацию ключевых процессов учета и управления.

Задачи

1. Провести анализ технического задания и определить требования к системе.
2. Разработать концептуальную, логическую и физическую модели базы данных.
3. Реализовать базу данных в СУБД SQLite.
4. Разработать пользовательский интерфейс с использованием Tkinter.
5. Реализовать функции добавления, редактирования, удаления и просмотра данных.
6. Реализовать функции формирования отчетов.
7. Провести тестирование и отладку системы.
8. Анализ технического задания.

В данной курсовой работе поставлена задача создания ИС для магазина покупки и продажи б/у телефонов. Для разработки такой системы необходимы СУБД для создания базы данных и среда разработки для создания интерфейса и функций по работе с БД.

Согласно заданию, в программе нужно учесть следующие особенности:

-Магазин скупает б/у телефоны, а затем продаёт их с наценкой.

-Так же магазин предоставляет услуги по обслуживанию телефонов и продажи аксессуаров для продаваемых телефонов.

- Магазин подсчитывает прибыль за определённый период времени для составления отчёта и отправки информации в налоговую.

Исходя из этих данных, БД должна содержать 3 таблицы, телефоны, аксессуары и услуги. Таблицы телефоны и аксессуары должны быть связанны т.к. магазин продаёт аксессуары только к продаваемым телефонам. Для разработки БД в рамках поставленной задачи можно использовать SQLite в качестве СУБД, и SQLite Studio в качестве графического приложения для отслеживания успешности работы в самом начале написания программного кода.

Достоинством SQLite является:

**Высокая скорость.** Благодаря особенностям архитектуры SQLite работает быстро, особенно на чтение. Компоненты СУБД встроены в приложение и вызываются в том же процессе. Поэтому доступ к ним быстрее, чем при взаимодействии между разными процессами.

**Хранение данных в одном файле.** База данных состоит из табличных записей, связей между ними, индексов и других компонентов. В SQLite они хранятся в едином файле (database file), который находится на том же устройстве, что и программа. Чтобы при работе не возникало ошибок, файл блокируется для сторонних процессов перед записью. Раньше это приводило к тому, что записывать данные в базу мог только один процесс единовременно. Но в новых версиях это решается перенастройкой режима работы СУБД.

**Минимализм.** Создатели SQLite пользуются принципом «минимального полного набора». Из всех возможностей SQL в ней есть наиболее нужные. Поэтому SQLite отличают малый размер, простота решений и легкость администрирования. Для повышения базовой функциональности можно использовать стороннее программное обеспечение и расширения.

**Надежность.** Код на 100% покрыт тестами. Это означает, что протестирован каждый компонент ПО. Поэтому SQLite считается надежной СУБД с минимальным риском непредсказуемого поведения.

**Нулевая конфигурация.** Перед использованием СУБД не нужна сложная настройка или длительная установка. Для решения большинства задач ей можно пользоваться «из коробки», без установки дополнительных компонентов.

**Малый размер.** Полностью сконфигурированный SQLite со всеми настройками занимает меньше 400 Кб. Если использовать СУБД без дополнительных компонентов, размер можно уменьшить до 250 Кб. Он зависит только от количества загруженной информации. Несмотря на малый размер, SQLite поддерживает большинство функций стандарта SQL2 и имеет ряд собственных.

**Доступность.**SQLite находится в публичном доступе. На ее использование нет правовых ограничений, а владельцем считается общество. Можно открывать, просматривать и изменять исходный код установленного ПО.

**Кроссплатформенность.** СУБД подходит для UNIX-подобных систем, MacOS и Windows.

**Автономность.** Система независима от стороннего ПО, библиотек или фреймворков. Чтобы приложение с базой на SQLite работало, дополнительные компоненты не требуются. Также не обязателен доступ в интернет: вся база хранится на устройстве, получить данные можно локально.

Для написания программы будет использоваться язык Python. Для разработки графической части будет использоваться библиотека tkinter.

Для того чтобы создать программу, необходимо учесть то, что она создается, прежде всего, для пользователя, и поэтому немаловажным требованием к программе должен стать удобный и интуитивно понятный интерфейс. Необходимо предусмотреть все возможности управления приложением, чтобы упростить работу пользователя и максимально обеспечить эффективность работы.

Программа должна правильно работать с данными, т.е всегда должен выводится нужный результат, требуемый пользователю. Приложение должно мгновенно реагировать на действия пользователя и в зависимости от запроса с его стороны формировать выходной результат.

**2 Разработка моделей данных**

Этот этап играет ключевую роль в разработке приложения для работы с базой данных. На этом этапе выделяются сущности, их атрибуты и взаимосвязи между ними. Создание и настройка базы данных в выбранной системе управления осуществляется на основе логической модели, которая затем преобразуется в физическую. Этот процесс включает в себя определение структуры таблиц, индексов, ключей и других объектов базы данных.

2.1 Концептуальная модель.

Создание логической модели начинается с анализа предметной области и выделения сущностей.

По заданию известно, что магазин покупает и продаёт б/у телефоны, предоставляет услуги по обслуживанию телефонов и продаёт аксессуары. Отсюда можно выделить три сущности – телефоны, аксессуары и услуги. Для сущности телефоны будут выделены следующие атрибуты – id, модель телефона, марка, IMEI телефона, дата покупки телефона магазином, дата продажи телефона, цена приёма телефона, цена продажи телефона, название процессора, кол-во оперативной памяти, название операционной системы. При приёме магазином телефона не указывается дата продажи. Для сущности аксессуары будут предоставлены атрибуты – id, telephone\_id, название аксессуара, цена и дата продажи. Атрибут telephone\_id является вторичным ключом, который ссылается на первичный ключ в сущности telephone(id). Для сущности услуги применимы следующие атрибуты – id, название услуги, цена, дата продажи. Сущность услуги не будет связанна ни с одной другой сущностью, т.к. услуги магазин может оказать любому телефону, неважно записан он в БД или нет, т.к. цену услуги определяет мастер.

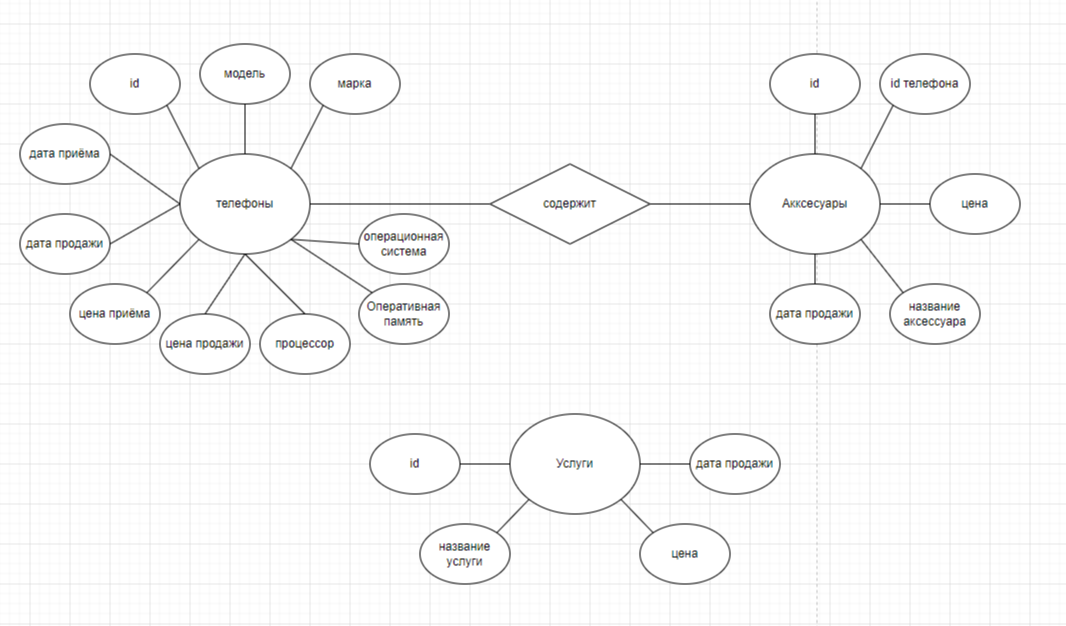


Рис1- концептуальная модель данных

**2.2. Разработка логической модели**

Разработка логической модели базы данных - это этап проектирования, на котором определяется структура и организация данных без учёта конкретной реализации в виде таблиц и столбцов.

Логическая модель базы данных описывает структуру данных, их атрибуты и взаимосвязи, но не затрагивает вопросы хранения информации. На этом этапе проектирования определяются сущности (таблицы), их атрибуты (столбцы) и связи между ними. Логическая модель позволяет понять, какие данные будут храниться, как они связаны и как будут использоваться. Этот этап предшествует разработке физической модели базы данных, которая представляет собой реальную реализацию в виде таблиц, ключей и связей.

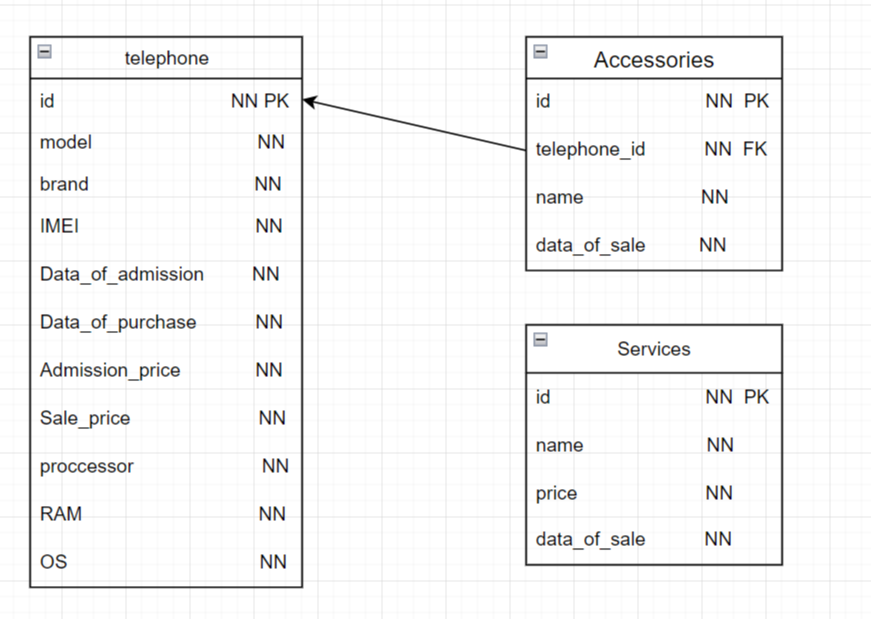


Рис 2- логическая модель данных

2.3. Разработка физической модели

Процесс разработки физической модели базы данных включает в себя несколько последовательных шагов, направленных на создание структуры для хранения и организации данных. Основные этапы этого процесса следующие:

1. Анализ требований: На этом этапе осуществляется сбор и анализ требований к базе данных, что включает определение сущностей и их атрибутов,

2. Физическая реализация: Здесь создается физическая структура базы данных, включая выбор типов данных, определение размеров полей и индексов, а также оптимизацию производительности.

3. Тестирование и оптимизация: После реализации физической модели базы данных проводится тестирование её функционирования и оптимизация для обеспечения эффективной работы.

* 1. Эти основные этапы помогают разработать физическую модель базы данных, которая будет эффективно хранить и обрабатывать данные.

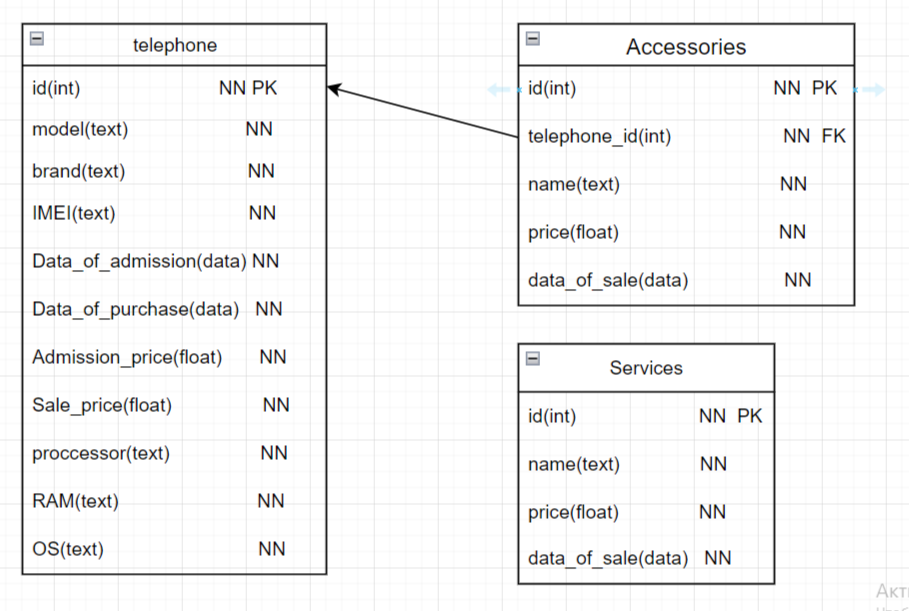


Рис 3 – физическая модель.

В результате разработки этих моделей достигается понимание оптимизации структуры данных и обеспечение эффективного хранения и обработки информации. Каждая модель играет свою роль в процессе создания базы данных, начиная с общего представления данных и заканчивая их конкретной реализацией.

3. Разработка и реализация ИС

3.1. Разработка средств защиты данных и программ.

Авторизация представляет собой процесс подтверждения личности пользователя и предоставления ему доступа к определённым ресурсам или функциям. В веб-разработке авторизация обычно осуществляется через ввод имени пользователя и пароля, которые используются для доступа к защищённым данным или выполнения определённых действий.

Аутентификация играет ключевую роль в обеспечении информационной безопасности и предотвращении несанкционированного доступа. Также возможно управлять правами доступа пользователей, определяя, какие ресурсы они могут использовать и какие действия выполнять в приложении.

Существует несколько методов авторизации, таких как базовая аутентификация, токены доступа и OAuth. Каждый из этих методов имеет свои преимущества и недостатки, и выбор конкретного метода зависит от потребностей приложения и необходимого уровня безопасности для защиты данных.

Тем не менее, внедрение авторизации может быть сложным процессом, требующим дополнительных ресурсов и усилий. Кроме того, некоторые приложения могут потребовать одобрения или противоречить своему основному назначению.

В целом, решение о внедрении авторизации в приложении должно приниматься с учётом потребностей пользователей, безопасности данных и целей проекта.

Преимущества авторизации:

1. Безопасность данных: Авторизация помогает предотвратить несанкционированный доступ к конфиденциальной информации.

2. Контроль доступа: Администраторы могут управлять правами пользователей и ограничивать функциональность приложения.

3. Улучшенный пользовательский опыт: Авторизация может обеспечить персонализированный опыт для пользователей, позволяя им сохранять свои настройки.

4. Предотвращение мошенничества: Она помогает избежать мошенничества, такого как взлом учётных записей и кража личных данных.

Недостатки авторизации:

1. Усложнение процесса входа: Добавление авторизации может сделать процесс входа более сложным для пользователей.

2. Проблемы с забытыми паролями: Пользователи могут забыть свои пароли и столкнуться с трудностями при доступе к приложению.

3. Расходы на поддержку: Предприятиям необходимо выделять дополнительные ресурсы на поддержку пользователей, испытывающих трудности с входом в систему.

4. Ограничения доступа: Авторизация может ограничить доступ некоторых пользователей к приложению, особенно если они сталкиваются с техническими проблемами или забытыми учётными данными.

Отказ от авторизации в моём приложении объясняется тем, что оно будет использоваться только пользователями, которые сидят за прилавком. В приложении не будет никакой конфиденциальной информации запрещённой к просмотру третьим лицам. Поэтому отсутствие авторизации упрощает использование приложения для целевой аудитории и улучшает общий пользовательский опыт. В конечном счёте, добавление авторизации может привести к функциональной избыточности и усложнению, что не соответствует целям и задачам приложения.

3.2. Разработка серверной части

На данном этапом нам нужно провести анализ требований к серверной части базы данных. Для её создания будет использоваться программное обеспечение SQLite Studio, в процессе проектирования которой, мы будем обращаться к разработанной нами раннее физической модели. В этой модели будут представлены описание всех сущностей, атрибуты и связей между ними.

Физическая модель базы данных описывает структуру и организацию данных на уровне их физического хранения. Она позволяет определить, как будет выглядеть наша база данных и как данные будут размещаться и храниться. Понимание этой концепции помогает разобраться в расположении данных в таблицах и установить связи между ними, что дает представление о готовой базе данных.

Кроме того, поддержка и обновление играют ключевую роль в обеспечении стабильной и безопасной работы информационной системы. Регулярные обновления серверной части базы данных помогают исправлять ошибки, устранять уязвимости и добавлять новые функции. Это способствует безопасности данных, а также повышает производительность и эффективность работы базы данных. Учитывая все вышеперечисленные аспекты, были разработаны следующие таблицы:

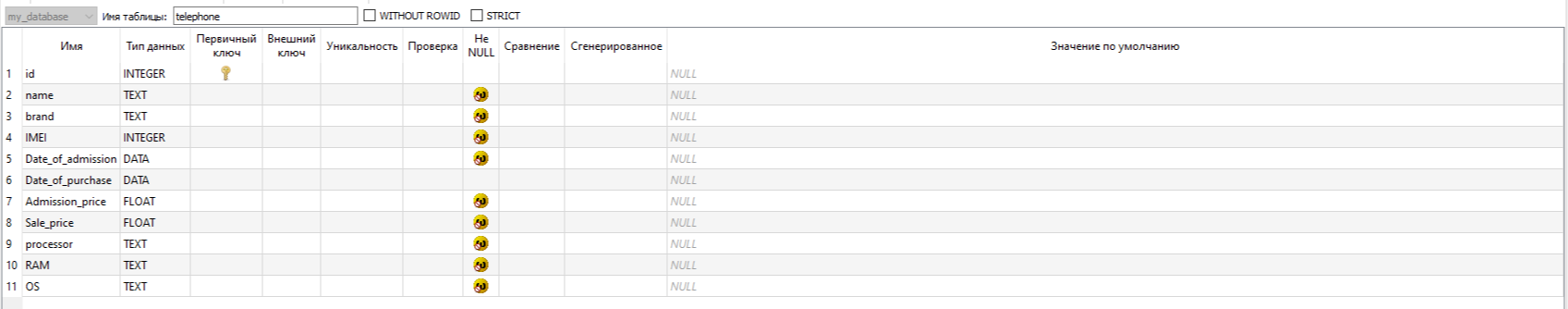


Рис 4.1- таблица телефонов SQLite Studio

Таблица 1 содержит информацию такую как ID телефона, модель, марку, IMEI, дату приёма, дату продажи, цену приёма, цену продажи, характеристики телефона



Рис 4.2 – таблица аксессуаров SQLite Studio.

Таблица 2 содержит информацию такую как ID аксессуара, ID телефона к которому продали аксессуар, название, цена, дата продажи.



Рис 4.3 – Таблица услуг SQLite Studio.

Таблица 3 содержит информацию такую как ID услуги, название, цена, дата продажи.

Разработка серверной части приложения для охранной организации завершилась успешно, из этого следует, что данная структура данных гарантирует надёжную основу для работы всего приложения.

3.3 Разработка SQL – запросов для ИС магазина покупки продажи б/у телефонов.

1. Для создания сущностей и атрибутов к ним используется запрос.  
   CREATE TABLE IF NOT EXISTS telephone (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

brand TEXT NOT NULL,

IMEI INTEGER NOT NULL,

Date\_of\_admission DATA NOT NULL,

Date\_of\_purchase DATA,

Admission\_price FLOAT NOT NULL,

Sale\_price FLOAT NOT NULL,

processor TEXT NOT NULL,

RAM TEXT NOT NULL,

OS TEXT NOT NULL

imafe\_path BLOB

)

CREATE TABLE IF NOT EXISTS Accessories (

id INTEGER PRIMARY KEY AUTOINCREMENT,

telephone\_id INTEGER,

name TEXT NOT NULL,

price FLOAT NOT NULL,

date\_of\_sale DATA NOT NULL,

FOREIGN KEY (telephone\_id) REFERENCES telephone(id)

)

CREATE TABLE IF NOT EXISTS Services (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

price FlOAT NOT NULL,

data\_of\_sale\_services DATA NOT NULL

)

1. Для получения данных о проданных телефонах используется SQL запрос.

SELECT \* FROM telephone WHERE Date\_of\_purchase IS NOT NULL

1. Для получения данных о не проданных телефонах используется SQL запрос.

SELECT \* FROM telephone WHERE Date\_of\_purchase IS NULL

1. Для получения прибыли по продаже телефонов используется запрос.

SELECT SUM(Sale\_price - Admission\_price) AS profit

FROM telephone

WHERE Date\_of\_purchase IS NOT NULL AND Date\_of\_purchase BETWEEN ? AND ?;

1. Это запрос для записи данных в таблицу telephone.

INSERT INTO telephone (name, brand, IMEI, Date\_of\_admission, Admission\_price, Sale\_price, processor, RAM, OS) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)', (\*values,)

1. Запрос для получения уникальных значений из столбце processor.

SELECT DISTINCT processor FROM telephone WHERE processor LIKE ?

1. Запрос для получения уникальных значений из столбца brand.

SELECT DISTINCT brand FROM telephone WHERE brand LIKE ?

1. Запрос для получения уникальных значений из столбца name.

SELECT DISTINCT name FROM telephone WHERE name LIKE ?

1. Запрос для получения всех данных из таблицы telephone.

SELECT \* FROM telephone

1. Запрос предназначенный для извлечения списка всех имен таблиц, существующих в текущей базе данных SQLite.

SELECT name FROM sqlite\_master WHERE type='table';

1. Запрос для получения данных из столбцов id, IMEI, brand, name из таблицы телефонов

SELECT id, IMEI, brand, name FROM telephone

1. Запрос предназначенный для занесения данных в таблицу аксессуаров.

'''

INSERT INTO Accessories (telephone\_id, name, price, date\_of\_sale)

VALUES (?, ?, ?, ?)''', (telephone\_id, name, price, date\_of\_sale)

1. Запрос предназначен для выборки данных из таблицы telephone на основе заданных критериев по марке (brand), модели (name), и IMEI, а затем сортирует результаты.

SELECT brand, name, IMEI, Date\_of\_purchase

FROM telephone

WHERE brand LIKE ? AND name LIKE ? AND IMEI LIKE ?

ORDER BY brand, name, IMEI

1. SQL-запрос предназначен для обновления даты покупки в таблице telephone для конкретного телефона, идентифицированного по его IMEI.

UPDATE telephone SET Date\_of\_purchase = ? WHERE IMEI = ?

1. Запрос предназначенный для получения уникальных значений из столбца марка (brand)

SELECT DISTINCT brand FROM telephone

1. Запрос предназначенный для получения уникальных значений из столбца name.

SELECT DISTINCT name FROM telephone

1. Запрос предназначенный для получения уникальных значений из столбца IMEI.

SELECT DISTINCT IMEI FROM telephone

1. SQL – запрос для записи данных в таблицу Services

'INSERT INTO Services (name, price, data\_of\_sale\_services) VALUES (?, ?, ?)',(service\_name, service\_price, sale\_date)

1. SQL-запрос для получения названий столбцов из выбранной пользователем таблицы.

PRAGMA table\_info({table\_name}) PRAGMA - это специальная команда в SQLite, которая используется для получения или изменения параметров базы данных. PRAGMA не является частью стандартного SQL, это специфическая команда SQLite.

1. Это - SQL запрос предназначен для обновления информации о записи в таблице telephone.

UPDATE telephone

SET name = ?, brand = ?, IMEI = ?, Date\_of\_admission = ?,

Date\_of\_purchase = ?, Admission\_price = ?, Sale\_price = ?, processor = ?, RAM = ?, OS = ?

WHERE id = ?""", (\*new\_values, values[0]))

1. Это - SQL запрос предназначен для обновления информации о записи в таблице Accessories.

UPDATE Accessories

SET telephone\_id = ?, name = ?, price = ?, date\_of\_sale = ?

WHERE id = ?""", (\*new\_values, values[0])

1. Это - SQL запрос предназначен для обновления информации о записи в таблице Services.

UPDATE Services

SET name = ?, price = ?, data\_of\_sale\_services = ?

WHERE id = ?""", (\*new\_values, values[0]))

1. SQL - запрос предназначенный для удаления записи из таблиц.

DELETE FROM {table\_combobox.get()} WHERE id = ?", (values[0],)

3.4. Разработка клиентской части

Интерфейс представляет собой способ взаимодействия между человеком и компьютерной системой, программой или устройством. Он включает в себя набор элементов, позволяющих пользователю взаимодействовать с приложением или устройством, выполнять определённые действия и получать нужную информацию.

Удобный интерфейс имеет решающее значение для пользовательского опыта, так как делает использование приложения более интуитивным и приятным. Важно, чтобы интерфейс был простым, понятным и легким в освоении, чтобы пользователи могли быстро и эффективно достигать своих целей. Хорошо продуманный интерфейс также помогает уменьшить количество ошибок и улучшает общее впечатление от приложения. Поэтому необходимо уделять особое внимание созданию интуитивно понятного и удобного интерфейса для обеспечения положительного опыта пользователей. При этом важно, чтобы интерфейс соответствовал необходимому функционалу, что также играет значительную роль в разработке приложения.

Исходя из всего вышесказанного, был разработан графический интерфейс пользователя:

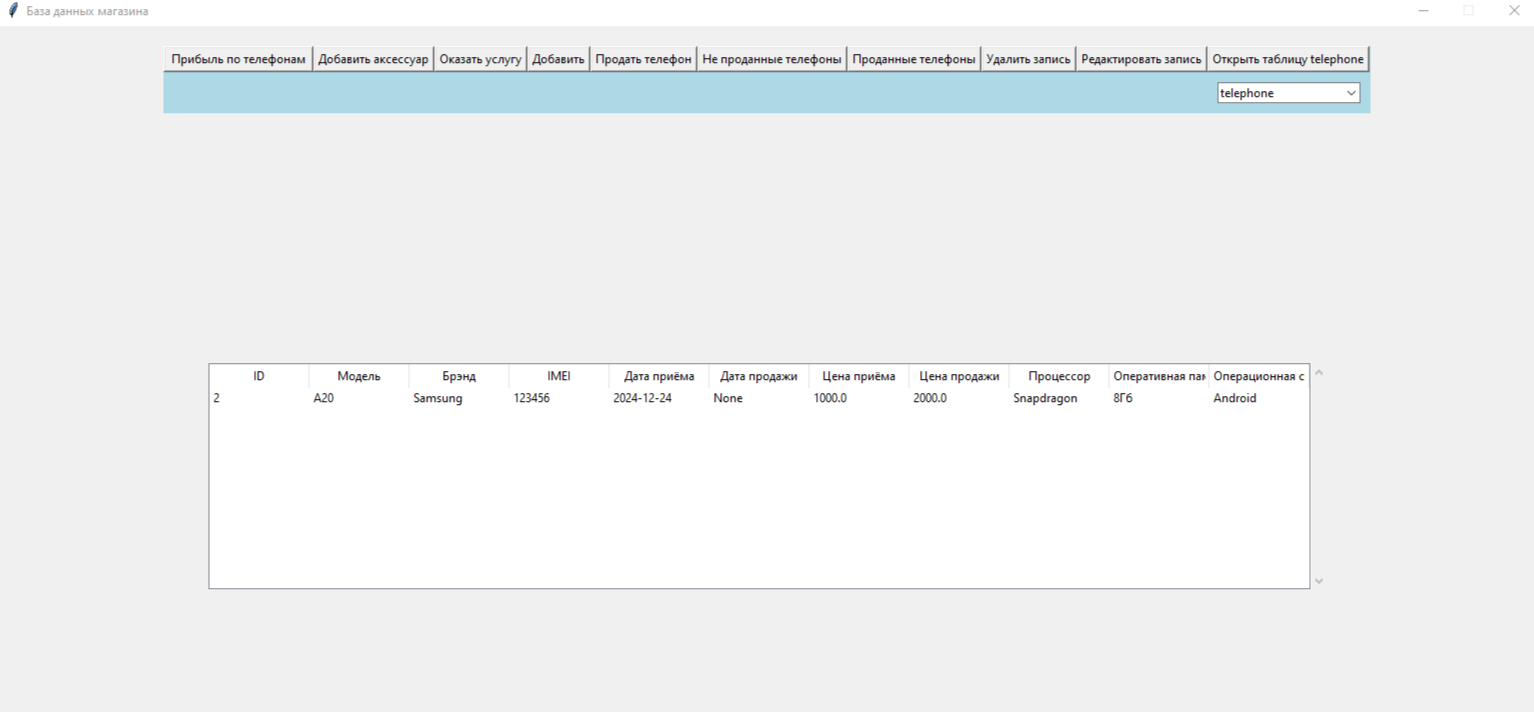


Рис 5.1 – главное окно приложения

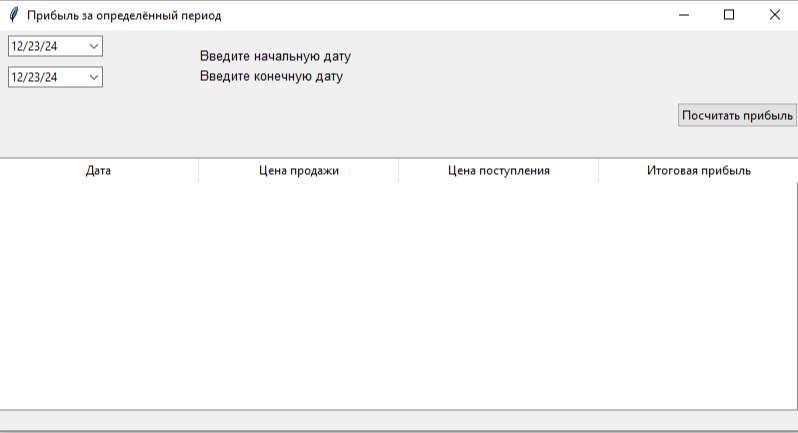


Рис 5.2 – интерфейс окна для вычисления прибыли за продажу телефонов

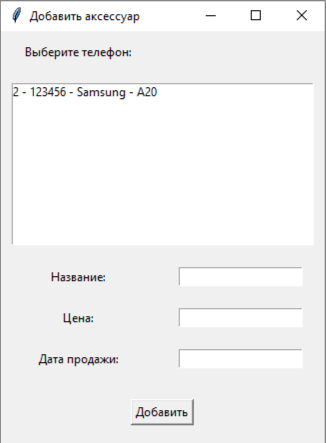


Рис 5.3 – интерфейс окно для добавления аксессуара для проданного телефона

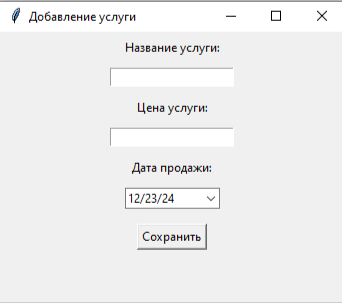


Рис 5.4 – интерфейс окна для добавления услуги

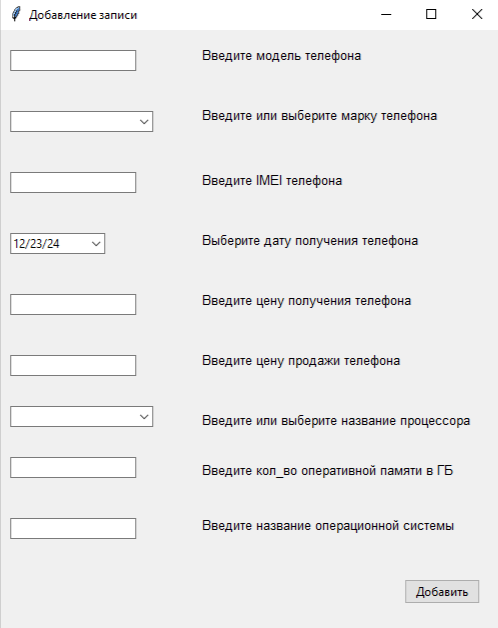


Рис 5.5 – интерфейс окна для добавления телефона.

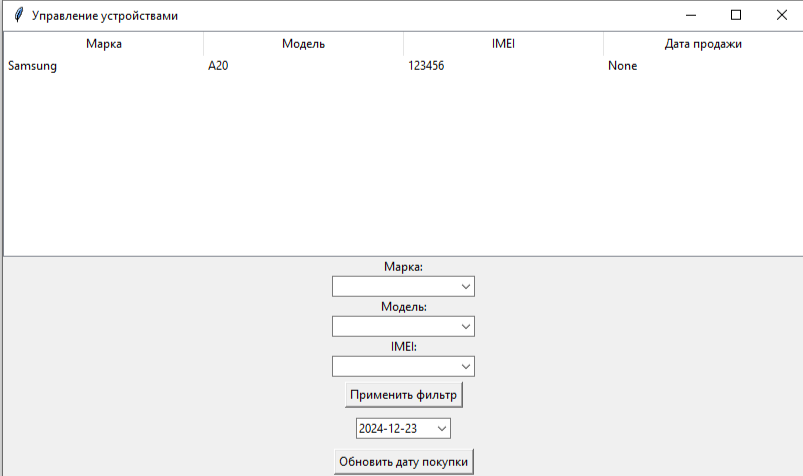


Рис 5.6 – интерфейс окна для продажи телефона.

При нажатии на кнопку “Продать телефон” откроется дополнительное окно где надо выбрать нужную запись из списка. В случае если записей слишком много и найти нужную будет невозможно, можно будет использовать фильтры для облегчения поиска нужной записи. При выборе записи надо будет указать в календаре дату продажи телефона

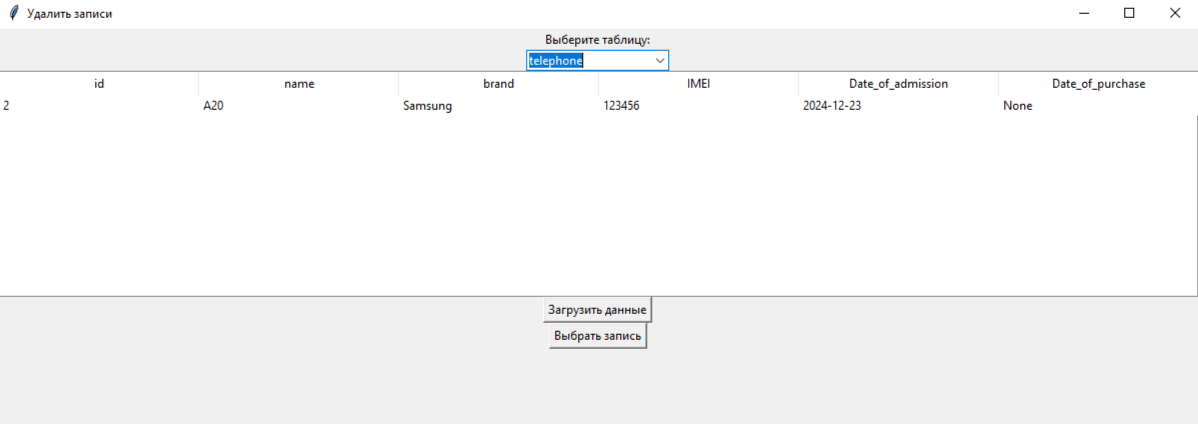


Рис 5.7 – интерфейс окна для удаления выбранной записи.

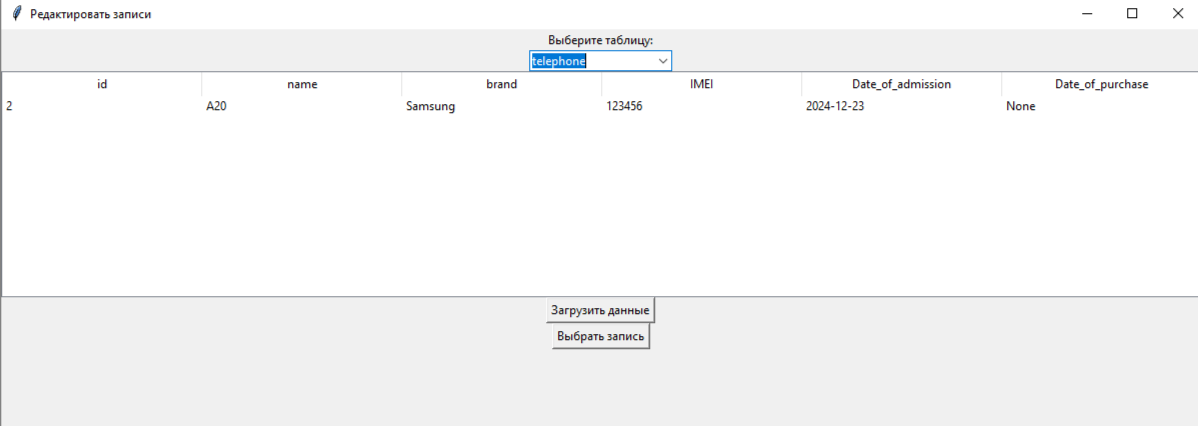


Рис 5.8 – интерфейс окна для редактирования выбранной записи.

3.5 Руководство пользователю и программисту:

1. Руководство программисту.

def open\_date\_window() – функция открытия дополнительного окна для вычисления прибыли по продажам телефонов.

def show\_sold\_items() – функция которая отображает в treeview записи телефонов, которые ещё не были проданы.

show\_sold\_items\_1() – функция которая отображает в treeview записи телефонов, которые были проданы магазином.

calculate\_profit\_and\_get\_records() – функция которая считает прибыль по продаже телефонов за определённый пользователем период времени.

def on\_calculate\_button\_click() – функция которая выводит в treeview записи из таблицы телефонов, которые подходят к промежутку времени указанного пользователем.

def open\_add\_window() – функция открытия дополнительного окна для добавления записи в таблицу telephone.

def add\_to\_db() – функция для добавления записи в таблицу telephone.

def fetch\_1\_data\_from\_db() – функция для получения уникальных значений из столбца processor и записи этих данных в список.

def tree\_populate\_combobox1() – функция для заполнения выпадающего списка combobox.

def tree\_on\_combobox\_changed() – функция для сравнений похожих значений из тех что ввёл пользователь и тех что записываются в выпадающий список combobox.

def fetch\_data(table\_names) – функция для получения данных из выбранной таблицы.

def on\_combobox\_select() – функция для создания treeview и заполнения его данными из выбранной пользователем таблицы.

def fetch\_telephones() – функция для получения данных из столбцов id, IMEI, brand, name и записи этих данных в список.

def insert\_accessory(telephone\_id, name, price, date\_of\_sale) – функция для записи данных в таблицу Accessory.

def add\_accessory() – функция для получения данных из окна “Добавить аксессуар” и проверки их на наполненность.

def on\_double\_click() – функция для двойного клика.

def open\_accessory\_window() – функция открытия дополнительного окна для добавления окна в таблицу accessory.

def load\_data(tree, filter\_brand="", filter\_model="", filter\_imei="") – функция для получения данных из столбцов name, model, imei.

def update\_purchase\_date(tree, date\_entry) – функция для обновления даты покупки и продаже телефонов.

def open\_new\_window() – функция открытия дополнительного окна для продажи телефонов.

def open\_add\_service\_window() – функция открытия дополнительного окна для продажи услуги.

def save\_service() – функция для сохранения данных в таблицу Service.

def edit\_or\_delete\_record(values, action) – функция для удаления или редактирования.

2.Руководство пользователю.

После запуска приложения откроется графический интерфейс. Он состоит из нескольких кнопок:

1. **Управление товарами**: - Для добавления нового телефона заполните соответствующие поля и нажмите кнопку "Добавить телефон", “Оказать услугу”, “Добавить аксессуар”.

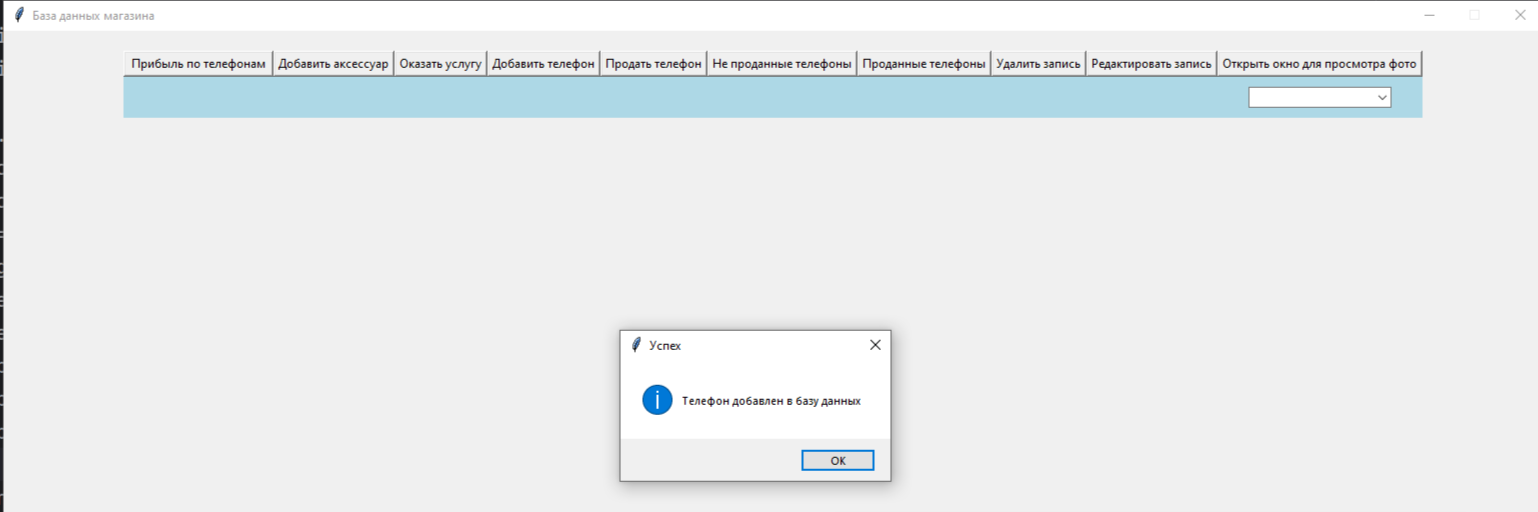
- Для удаления телефона, аксессуара, услуги нажмите на кнопку “Удалить запись” выберите нужную таблицу из списка и нажмите "Загрузить данные". После выберите нужную запись для удаления и нажмите кнопку “Выбрать запись”.

- Для обновления информации о телефоне, аксессуаре или услуге нажмите на кнопку “Редактировать запись” после в выпадающем списке выберите нужную таблицу и нажмите кнопку “Загрузить данные”. После выберите нужную запись и нажмите кнопку ”Выбрать запись”. Появятся данные из выбранной записи которые можно будет изменить.

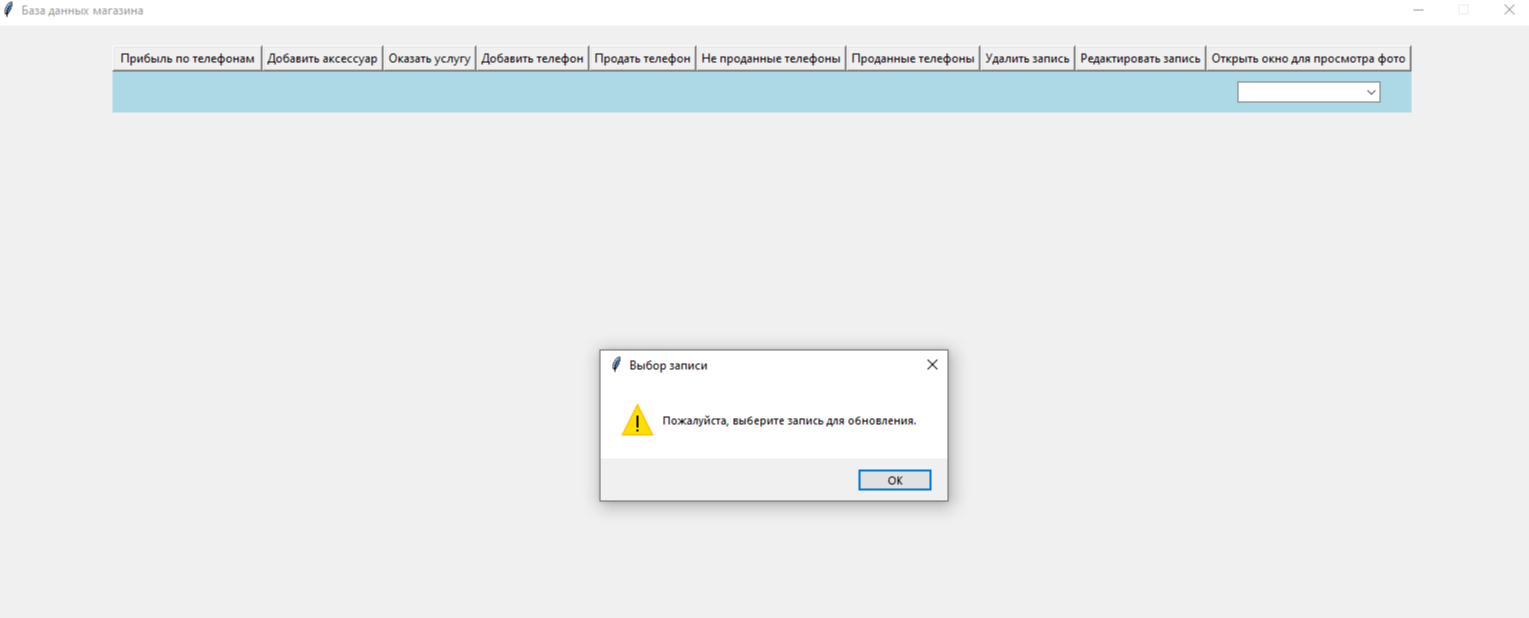
. - Для просмотра списка телефонов просто прокрутите поле списка или воспользуйтесь функцией поиска в окне которое открывается с кнопки “Продать телефон”.

4. Тестирование ИС.

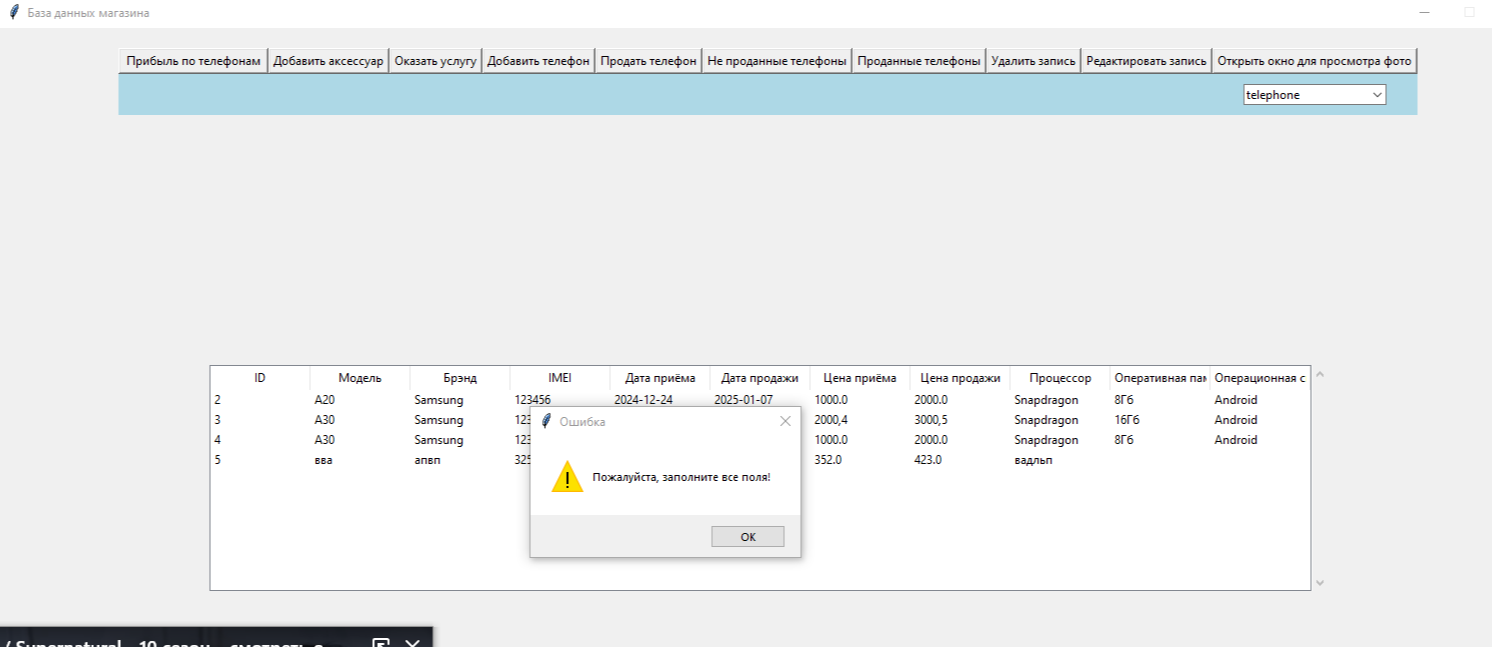
При завершении разработки программы необходимо провести проверку на её работоспособность при различных действиях пользователей. Программа должна корректно отображать и выполнять действия пользователя в любом случае. В исключительных моментах должны выводиться окна с сообщениями, поясняющие ситуацию.

При добавлении записи появляется сообщение об удачном добавлении телефона в БД. Если пользователь не занесет, какие либо данные в окно “Добавление записи” то запись всё равно добавится т.к. на старых телефонах не всегда можно узнать некоторые данные.

При продаже телефона пользователь может использовать разные фильтры, но если пользователь не выберет ни одну запись, то появится уведомление о том что нужно обязательно выбрать запись.



При оказании услуги если не заполнены все поля то появится предупреждение о том что заполнить оставшиеся поля.



Заключение:

В ходе выполнения курсовой работы по теме "Информационная система магазина продажи телефонов" была разработана полноценная информационная система, отвечающая современным требованиям торговли и обслуживания мобильных устройств. Систематизация данных о продаже телефонов бывших в употреблении и новых аксессуарах позволила оптимизировать процессы учета и анализа. База данных включает в себя ключевую информацию, такую как марка и модель телефона, его основные характеристики, а также цены на прием и продажу, что значительно упрощает ведение бизнеса. Кроме того, внедрение возможности хранения изображений телефонов и аксессуаров в базе данных добавляет визуальную составляющую к продукции, что способствует более наглядному представлению информации для сотрудников и клиентов магазина. В целом, разработанная информационная система является надежным инструментом, который улучшит процессы учета, анализа и предоставления услуг в магазине, обеспечивая высокий уровень сервиса и удовлетворение потребностей клиентов. Подобная система способна существенно повысить конкурентоспособность магазина на рынке реализации мобильных устройств и сопутствующих товаров.

Список литературы

1. **Кормен Т. Х., Лейзерсон Ч. E., Ривест Р. L., Штайн К.** Введение в алгоритмы. - М.: Вильямс, 2017.

2. **Мадьяров М. А.** проектирование баз данных: Учебное пособие. - СПб.: БХВ-Петербург, 2020.

3. **Бурлоцкий И. А.**. SQL для профессионалов. - М.: ДМК Press, 2019.

4. **Китаева Т. Г.**. Основы баз данных: концепции, технологии, приложения. - М.: Кнорус, 2021.

Приложение 1 Модели данных.

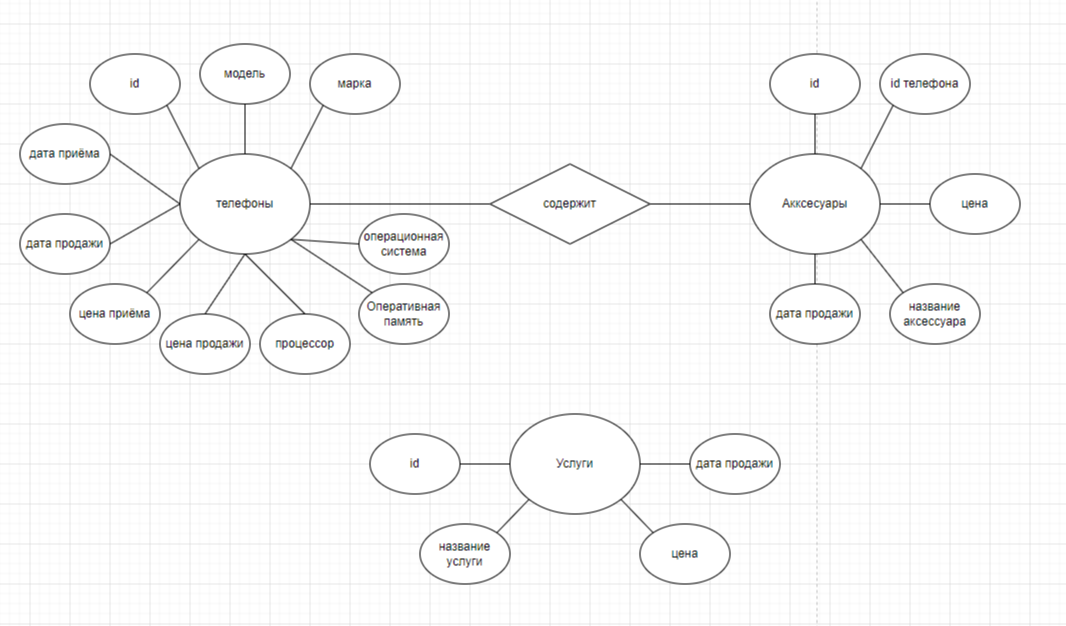


Рис1 – концептуальная модель данных

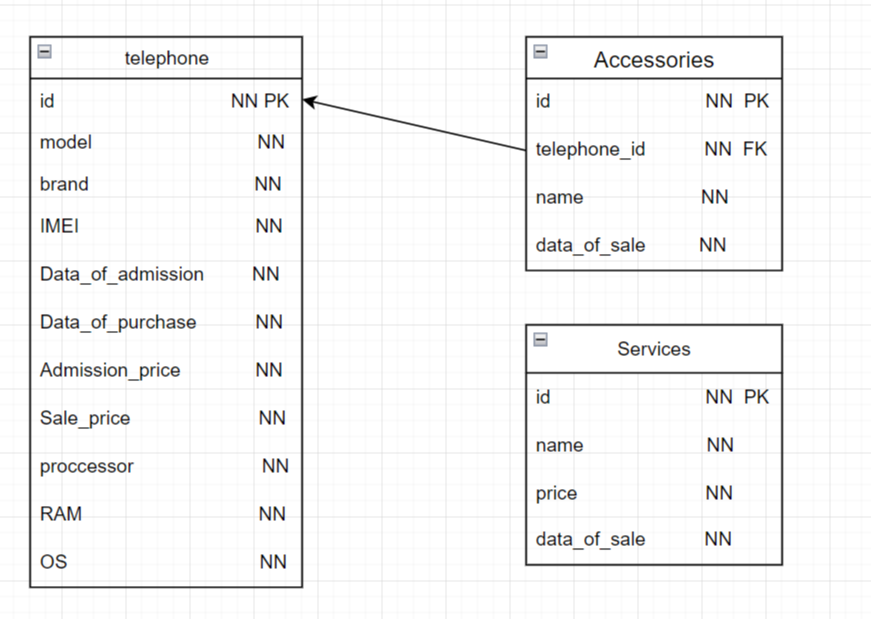


Рис2 -логическая модель данных

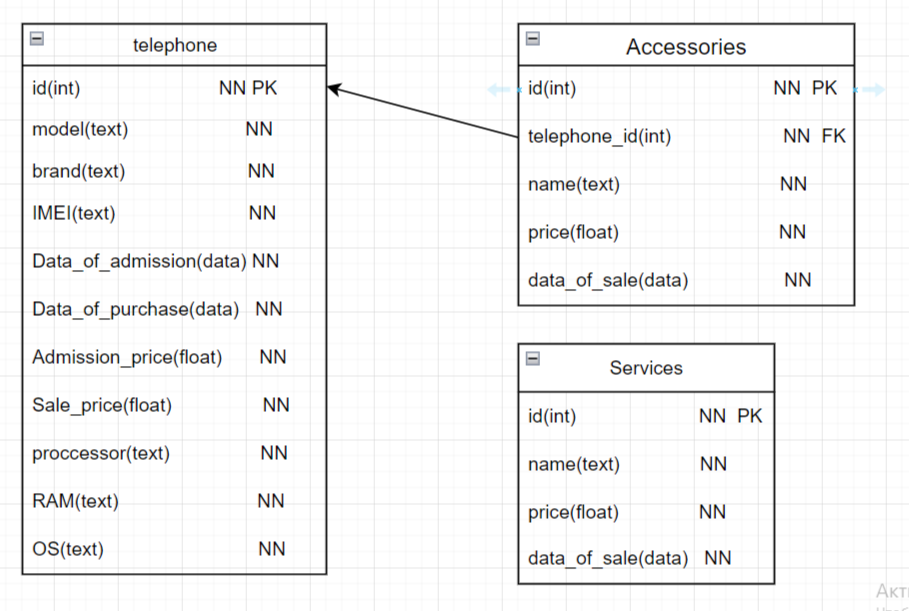


Рис3 - физическая модель данных

Приложение 2 Текст программы.

import sqlite3

from tkinter import \*

import tkinter as tk

from tkinter import ttk, filedialog

from tkinter.ttk import Combobox

from tkcalendar import DateEntry

from tkinter import messagebox

from datetime import datetime

from PIL import ImageTk, Image

connection = sqlite3.connect('my\_database.db') #Подключаемся к базе данных

cursor = connection.cursor()

connection.execute("PRAGMA foreign\_keys = ON;")

cursor.execute('''

CREATE TABLE IF NOT EXISTS telephone (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

brand TEXT NOT NULL,

IMEI INTEGER NOT NULL,

Date\_of\_admission DATA NOT NULL,

Date\_of\_purchase DATA,

Admission\_price FLOAT NOT NULL,

Sale\_price FLOAT NOT NULL,

processor TEXT NOT NULL,

RAM TEXT NOT NULL,

OS TEXT NOT NULL,

image\_path BLOB

)

''')

connection1 = sqlite3.connect('my\_database.db') #Подключаемся к базе данных

cursor1 = connection1.cursor()

cursor1.execute('''

CREATE TABLE IF NOT EXISTS Accessories (

id INTEGER PRIMARY KEY AUTOINCREMENT,

telephone\_id INTEGER,

name TEXT NOT NULL,

price FLOAT NOT NULL,

date\_of\_sale DATA NOT NULL,

FOREIGN KEY (telephone\_id) REFERENCES telephone(id)

)

''')

connection2 = sqlite3.connect('my\_database.db') #Подключаемся к базе данных

cursor5 = connection2.cursor()

cursor5.execute('''

CREATE TABLE IF NOT EXISTS Services (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

price FlOAT NOT NULL,

data\_of\_sale\_services DATA NOT NULL

)

''')

# Функция для обновления Treeview

def show\_sold\_items():

# Очищаем текущие записи в Treeview

for row in tree.get\_children():

tree.delete(row)

# Подключаемся к базе данных

try:

connection = sqlite3.connect("my\_database.db") # Замените на имя вашей базы данных

cursor = connection.cursor()

# Выполняем запрос для получения проданных товаров

cursor.execute("SELECT \* FROM telephone WHERE Date\_of\_purchase IS NOT NULL")

sold\_items = cursor.fetchall()

# Проверяем, есть ли записи

if not sold\_items:

messagebox.showinfo("Информация", "Нет проданных товаров с датой покупки.")

return

# Вставляем записи в Treeview

for item in sold\_items:

tree.insert("", "end", values=item)

except sqlite3.Error as e:

messagebox.showerror("Ошибка базы данных", str(e))

finally:

if connection:

connection.close()

def show\_sold\_items\_1():

# Очищаем текущие записи в Treeview

for row in tree.get\_children():

tree.delete(row)

# Подключаемся к базе данных

try:

connection = sqlite3.connect("my\_database.db") # Замените на имя вашей базы данных

cursor = connection.cursor()

# Выполняем запрос для получения проданных товаров

cursor.execute("SELECT \* FROM telephone WHERE Date\_of\_purchase IS NULL")

sold\_items = cursor.fetchall()

# Проверяем, есть ли записи

if not sold\_items:

messagebox.showinfo("Информация", "Нет купленных товаров.")

return

# Вставляем записи в Treeview

for item in sold\_items:

tree.insert("", "end", values=item)

except sqlite3.Error as e:

messagebox.showerror("Ошибка базы данных", str(e))

finally:

if connection:

connection.close()

def open\_date\_window():

add\_window\_2 = tk.Toplevel(root)

add\_window\_2.title("Прибыль за определённый период")

add\_window\_2.geometry("800x400")

global start\_date\_entry

start\_date\_entry = DateEntry(add\_window\_2,width=12, background='darkblue', foreground='white', borderwidth=2, pady=20)

start\_date\_entry.pack(pady=5, anchor="nw", padx=10)

global end\_date\_entry

end\_date\_entry = DateEntry(add\_window\_2, background='darkblue', foreground='white', borderwidth=2, pady=20)

end\_date\_entry.pack(pady=5, anchor="nw", padx=10)

label1 = ttk.Label(add\_window\_2, text="Введите начальную дату", font=("Arial", 10))

label1.place(x=200, y=15)

label2 = ttk.Label(add\_window\_2, text="Введите конечную дату", font=("Arial", 10))

label2.place(x=200, y=35)

calculate\_button = ttk.Button(add\_window\_2, text="Посчитать прибыль", command=on\_calculate\_button\_click)

calculate\_button.pack(pady=10, anchor="se")

# Создание Treeview для отображения записей

global treeview

treeview = ttk.Treeview(add\_window\_2, columns=("Date", "Sale Price", "Admission Price", "Total Profit"),

show="headings")

treeview.heading("Date", text="Дата")

treeview.heading("Sale Price", text="Цена продажи")

treeview.heading("Admission Price", text="Цена поступления")

treeview.heading("Total Profit", text="Итоговая прибыль")

treeview.pack(pady=20, fill=tk.BOTH, expand=True)

def calculate\_profit\_and\_get\_records(start\_date, end\_date):

conn = sqlite3.connect('my\_database.db')

cursor = conn.cursor()

profit\_query = """

SELECT SUM(Sale\_price - Admission\_price) AS profit

FROM telephone

WHERE Date\_of\_purchase IS NOT NULL AND Date\_of\_purchase BETWEEN ? AND ?;

"""

cursor.execute(profit\_query, (start\_date, end\_date))

profit = cursor.fetchone()[0] # Здесь profit может быть None, если нет записей

records\_query = """

SELECT Date\_of\_admission, Sale\_price, Admission\_price

FROM telephone

WHERE Date\_of\_admission BETWEEN ? AND ?;

"""

cursor.execute(records\_query, (start\_date, end\_date))

records = cursor.fetchall()

conn.close()

# Возвращаем profit, даже если он отрицательный, и records

return profit, records

def on\_calculate\_button\_click():

start\_date\_time = start\_date\_entry.get\_date()

start\_date = start\_date\_time.strftime('%Y-%m-%d')

end\_date\_time = end\_date\_entry.get\_date()

end\_date = end\_date\_time.strftime('%Y-%m-%d')

if not start\_date or not end\_date:

tk.messagebox.showerror("Ошибка", "Пожалуйста, введите обе даты.")

return

# Вычисление прибыли и получение записей

profit, records = calculate\_profit\_and\_get\_records(start\_date, end\_date)

# Очистка Treeview перед отображением новых данных

for row in treeview.get\_children():

treeview.delete(row)

# Добавление записей в Treeview

for record in records:

treeview.insert("", tk.END, values=record + (None,))

# Добавление строки с итоговой прибылью

total\_profit\_record = (None, None, None, profit) # profit может быть отрицательным

treeview.insert("", tk.END, values=total\_profit\_record)

def open\_add\_window():

add\_window = tk.Toplevel(root)

add\_window.title("Добавление записи")

add\_window.geometry("500x600")

entry = ttk.Entry(add\_window)

entry.pack(pady=20, anchor="nw", padx=10)

combobox = ttk.Combobox(add\_window)

combobox.pack(pady=20, anchor="nw", padx=10)

entry1 = ttk.Entry(add\_window)

entry1.pack(pady=20, anchor="nw", padx=10)

date\_entry = DateEntry(add\_window, width=12, background='darkblue', foreground='white', borderwidth=2, pady=20)

date\_entry.pack(pady=20, anchor="nw", padx=10)

entry2 = ttk.Entry(add\_window)

entry2.pack(pady=20, anchor="nw", padx=10)

entry3 = ttk.Entry(add\_window)

entry3.pack(pady=20, anchor="nw", padx=10)

combobox2 = ttk.Combobox(add\_window)

combobox2.pack(pady=10, anchor="nw", padx=10)

entry4 = ttk.Entry(add\_window)

entry4.pack(pady=20, anchor="nw", padx=10)

entry5 = ttk.Entry(add\_window)

entry5.pack(pady=20, anchor="nw", padx=10)

label = ttk.Label(add\_window, text="Введите модель телефона", font=("Arial", 10))

label.place(x=200,y=15)

label1 = ttk.Label(add\_window, text="Введите или выберите марку телефона", font=("Arial", 10))

label1.place(x=200, y=75)

label2 = ttk.Label(add\_window, text="Введите IMEI телефона", font=("Arial", 10))

label2.place(x=200, y=140)

label3 = ttk.Label(add\_window, text="Выберите дату получения телефона", font=("Arial", 10))

label3.place(x=200, y=200)

label4 = ttk.Label(add\_window, text="Введите цену получения телефона", font=("Arial", 10))

label4.place(x=200, y=260)

label5 = ttk.Label(add\_window, text="Введите цену продажи телефона", font=("Arial", 10))

label5.place(x=200, y=320)

label6 = ttk.Label(add\_window, text="Введите или выберите название процессора", font=("Arial", 10))

label6.place(x=200, y=380)

label7 = ttk.Label(add\_window, text="Введите кол\_во оперативной памяти в ГБ", font=("Arial", 10))

label7.place(x=200, y=430)

label8 = ttk.Label(add\_window, text="Введите название операционной системы", font=("Arial", 10))

label8.place(x=200, y=485)

def add\_to\_db():

date\_admission = date\_entry.get\_date() # Получаем объект datetime.date

date\_admission\_str = date\_admission.strftime('%Y-%m-%d')

values = [entry.get(), combobox.get(), entry1.get(), date\_admission\_str, entry2.get(), entry3.get(), combobox2.get(), entry4.get(), entry5.get()]

conn = sqlite3.connect('my\_database.db')

cursor = conn.cursor()

cursor.execute('INSERT INTO telephone (name, brand, IMEI, Date\_of\_admission, Admission\_price, Sale\_price, processor, RAM, OS) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)', (\*values,))

conn.commit()

conn.close()

entry.delete(0, tk.END) # Очистка поля ввода

combobox.delete(0, tk.END)

entry1.delete(0, tk.END)

entry2.delete(0, tk.END)

entry3.delete(0, tk.END)

entry4.delete(0, tk.END)

entry5.delete(0, tk.END)

combobox2.delete(0,tk.END)

combobox.set('') # Сброс выбора в Combobox

messagebox.showinfo("Успех", "Телефон добавлен в базу данных")

button1 = ttk.Button(add\_window, text="Добавить", command=add\_to\_db)

button1.pack(pady=20, padx=20, anchor="se")

def fetch\_1\_data\_from\_db(prefix):

# Подключаемся к базе данных

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

# Выполняем SQL-запрос с LIKE для поиска по первым буквам

cursor.execute("SELECT DISTINCT processor FROM telephone WHERE processor LIKE ?",

(prefix + '%',))

rowss = cursor.fetchall()

# Закрываем соединение с базой данных

connection.close()

# Возвращаем уникальные значения в виде списка

return [row[0] for row in rowss]

def tree\_populate\_combobox1():

values = fetch\_1\_data\_from\_db("")

combobox2['values'] = values

def tree\_on\_combobox\_changed(event):

current1\_text = combobox2.get()

if current1\_text:

values = get\_data\_from\_db(current1\_text)

combobox['values'] = values

if values:

combobox2.focus\_set() # Устанавливаем фокус обратно на combobox

combobox2.event\_generate('<Down>') # Открываем выпадающий список

def fetch\_data\_from\_db(prefix):

# Подключаемся к базе данных

connection = sqlite3.connect('my\_database.db') # Замените на ваше имя базы данных

cursor = connection.cursor()

# Выполняем SQL-запрос с LIKE для поиска по первым буквам

cursor.execute("SELECT DISTINCT brand FROM telephone WHERE brand LIKE ?",

(prefix + '%',))

rowss = cursor.fetchall()

# Закрываем соединение с базой данных

connection.close()

# Возвращаем уникальные значения в виде списка

return [row[0] for row in rowss]

def get\_data\_from\_db(prefix):

# Подключаемся к базе данных

connection = sqlite3.connect('my\_database.db') # Замените на ваше имя базы данных

cursor = connection.cursor()

# Выполняем SQL-запрос с LIKE для поиска по первым буквам

cursor.execute("SELECT DISTINCT name FROM telephone WHERE name LIKE ?",

(prefix + '%',)) # Замените your\_column и your\_table на ваши названия

rows = cursor.fetchall()

# Закрываем соединение с базой данных

connection.close()

# Возвращаем уникальные значения в виде списка

return [row[0] for row in rows]

def populate\_combobox():

# Функция первоначального заполнения

values = fetch\_data\_from\_db("") # Получаем все значения для начального заполнения

combobox['values'] = values

def two\_populate\_combobox1():

values = get\_data\_from\_db("")

combobox1['values'] = values

def two\_on\_combobox\_changed(event):

current1\_text = combobox1.get()

if current1\_text:

values = get\_data\_from\_db(current1\_text)

combobox['values'] = values

if values:

combobox1.focus\_set() # Устанавливаем фокус обратно на combobox

combobox1.event\_generate('<Down>') # Открываем выпадающий список

def on\_combobox\_changed(event):

# Получаем текущее значение ввода из Combobox

current\_text = combobox.get()

if current\_text:

# Получаем обновленные значения из базы данных

values = fetch\_data\_from\_db(current\_text)

combobox['values'] = values

# Открываем выпадающий список, чтобы пользователь мог выбрать из новых значений

if values:

combobox.focus\_set() # Устанавливаем фокус обратно на combobox

combobox.event\_generate('<Down>') # Открываем выпадающий список

combobox.bind("<KeyRelease>", on\_combobox\_changed)

#combobox1.bind("<KeyRelease>", two\_on\_combobox\_changed)

combobox2.bind("<KeyRelease>", tree\_on\_combobox\_changed)

populate\_combobox()

#two\_populate\_combobox1()

tree\_populate\_combobox1()

# Прописываем названия колонок для каждой таблицы

columns\_mapping = {

"telephone": ["ID", "Модель", "Брэнд", "IMEI", "Дата приёма", "Дата продажи", "Цена приёма", "Цена продажи", "Процессор", "Оперативная память", "Операционная система"],

"Services": ["ID", "Название услуги", "Цена", "Дата продажи"],

"Accessories": ["ID", "id\_Телефона", "Название аксессуара", "цена", "Дата продажи"]

}

def fetch\_data(table\_names):

# Соединяемся с базой данных

conn = sqlite3.connect('my\_database.db')

cursor = conn.cursor()

# Получаем данные из выбранной таблицы

cursor.execute(f"SELECT \* FROM {table\_names}")

data = cursor.fetchall()

conn.close()

return data

def on\_combobox\_select(event):

# Удаляем старый Treeview, если он существует

for widget in treeview\_frame.winfo\_children():

widget.destroy()

# Получаем выбранную таблицу

selected\_table = combobox.get()

# Извлекаем данные и названия столбцов

data = fetch\_data(selected\_table)

columns = columns\_mapping[selected\_table] # Используем словарь для названий столбцов

# Создаем новый Treeview

global tree

tree = ttk.Treeview(treeview\_frame, show="headings")

tree["columns"] = columns

# Установка заголовков столбцов

for col in columns:

tree.heading(col, text=col)

# Заполнение Treeview данными

for item in data:

tree.insert("", tk.END, values=item)

# Установка ширины столбцов

for col in columns:

tree.column(col, width=100) # Установите ширину по желанию

# Размещение нового Treeview

tree.pack(fill=tk.BOTH, expand=True)

# Добавление вертикальной полосы прокрутки

vertical\_scrollbar = ttk.Scrollbar(treeview\_frame, orient="vertical", command=tree.yview)

tree.configure(yscrollcommand=vertical\_scrollbar.set)

vertical\_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)

tree.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Подключаемся к базе данных и извлекаем названия таблиц

conn = sqlite3.connect('my\_database.db')

cursor = conn.cursor()

cursor.execute("SELECT name FROM sqlite\_master WHERE type='table';")

table\_names = [row[0] for row in cursor.fetchall()]

conn.close()

# Функция для извлечения данных из базы данных

def fetch\_telephones():

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

cursor.execute('SELECT id, IMEI, brand, name FROM telephone')

rows = cursor.fetchall()

connection.close()

return rows

def insert\_accessory(telephone\_id, name, price, date\_of\_sale):

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

cursor.execute('''

INSERT INTO Accessories (telephone\_id, name, price, date\_of\_sale)

VALUES (?, ?, ?, ?)

''', (telephone\_id, name, price, date\_of\_sale))

connection.commit()

connection.close()

def add\_accessory():

telephone\_id = selected\_telephone\_id.get()

name = name\_entry.get()

price = price\_entry.get()

date\_of\_sale = date\_entry.get()

# Проверка на заполненность полей

if not all([telephone\_id, name, price, date\_of\_sale]):

messagebox.showwarning("Ввод данных", "Пожалуйста, заполните все поля.")

return

insert\_accessory(telephone\_id, name, price, date\_of\_sale)

messagebox.showinfo("Успех", "Аксессуар успешно добавлен!")

accessory\_window.destroy()

def on\_double\_click(event):

selected = telephone\_listbox.curselection()

if selected:

index = selected[0]

selected\_item = telephone\_listbox.get(index)

telephone\_id = selected\_item.split(" - ")[0] # Получаем только id

selected\_telephone\_id.set(telephone\_id)

def open\_accessory\_window():

global accessory\_window, name\_entry, price\_entry, date\_entry, selected\_telephone\_id, telephone\_listbox

accessory\_window = tk.Toplevel(root)

accessory\_window.title("Добавить аксессуар")

tk.Label(accessory\_window, text="Выберите телефон:").grid(row=0, column=0, padx=10, pady=10)

selected\_telephone\_id = tk.StringVar()

telephone\_listbox = tk.Listbox(accessory\_window, width=50)

telephone\_listbox.grid(row=1, column=0, padx=10, pady=10, columnspan=2)

# Заполнение списка телефонами

for id, imei, brand, model in fetch\_telephones():

telephone\_listbox.insert(tk.END, f"{id} - {imei} - {brand} - {model}")

telephone\_listbox.bind("<Double-1>", on\_double\_click) # Привязываем двойной клик

tk.Label(accessory\_window, text="Название:").grid(row=2, column=0, padx=10, pady=10)

name\_entry = tk.Entry(accessory\_window)

name\_entry.grid(row=2, column=1, padx=10, pady=10)

tk.Label(accessory\_window, text="Цена:").grid(row=3, column=0, padx=10, pady=10)

price\_entry = tk.Entry(accessory\_window)

price\_entry.grid(row=3, column=1, padx=10, pady=10)

tk.Label(accessory\_window, text="Дата продажи:").grid(row=4, column=0, padx=10, pady=10)

date\_entry = tk.Entry(accessory\_window)

date\_entry.grid(row=4, column=1, padx=10, pady=10)

tk.Button(accessory\_window, text="Добавить", command=add\_accessory).grid(row=5, column=0, columnspan=2, pady=20)

def load\_data(tree, filter\_brand="", filter\_model="", filter\_imei=""):

for item in tree.get\_children():

tree.delete(item)

connection = sqlite3.connect('my\_database.db')

cursor2 = connection.cursor()

query = """

SELECT brand, name, IMEI, Date\_of\_purchase

FROM telephone

WHERE brand LIKE ? AND name LIKE ? AND IMEI LIKE ?

ORDER BY brand, name, IMEI

"""

cursor2.execute(query, (f"%{filter\_brand}%", f"%{filter\_model}%", f"%{filter\_imei}%"))

for row in cursor2.fetchall():

tree.insert("", tk.END, values=row)

connection.close()

def on\_item\_select(event, tree, date\_entry):

selected\_item = tree.selection()

if selected\_item:

item\_values = tree.item(selected\_item[0], 'values')

date\_entry.delete(0, tk.END)

date\_entry.insert(0, item\_values[3]) # Вставляем дату покупки в поле

def update\_purchase\_date(tree, date\_entry):

selected\_item = tree.selection()

if not selected\_item:

messagebox.showwarning("Выбор записи", "Пожалуйста, выберите запись для обновления.")

return

new\_date = date\_entry.get()

item\_values = tree.item(selected\_item[0], 'values')

imei = item\_values[2]

connection1 = sqlite3.connect('my\_database.db')

cursor1 = connection1.cursor()

cursor1.execute("UPDATE telephone SET Date\_of\_purchase = ? WHERE IMEI = ?", (new\_date, imei))

connection1.commit()

connection1.close()

load\_data(tree) # Обновляем данные в дереве

messagebox.showinfo("Успех", "Данные продажи обновленны")

def load\_combobox\_data():

connection3 = sqlite3.connect('my\_database.db')

cursor3 = connection3.cursor()

cursor3.execute("SELECT DISTINCT brand FROM telephone")

brands = [row[0] for row in cursor3.fetchall()]

cursor3.execute("SELECT DISTINCT name FROM telephone")

models = [row[0] for row in cursor3.fetchall()]

cursor3.execute("SELECT DISTINCT IMEI FROM telephone")

imeis = [row[0] for row in cursor3.fetchall()]

connection3.close()

return brands, models, imeis

def filter\_data(tree, brand\_combobox, model\_combobox, imei\_combobox):

filter\_brand = brand\_combobox.get()

filter\_model = model\_combobox.get()

filter\_imei = imei\_combobox.get()

load\_data(tree, filter\_brand, filter\_model, filter\_imei)

def open\_new\_window():

new\_window = tk.Toplevel(root)

new\_window.title("Управление устройствами")

global tree

tree = ttk.Treeview(new\_window, columns=("Brand", "Model", "IMEI", "Date of Purchase"), show='headings')

tree.heading("Brand", text="Марка")

tree.heading("Model", text="Модель")

tree.heading("IMEI", text="IMEI")

tree.heading("Date of Purchase", text="Дата продажи")

tree.pack(fill=tk.BOTH, expand=True)

load\_data(tree)

# Загружаем данные для комбобоксов

brands, models, imeis = load\_combobox\_data()

brand\_label = ttk.Label(new\_window, text="Марка:")

brand\_label.pack()

brand\_combobox = ttk.Combobox(new\_window, values=brands)

brand\_combobox.pack()

model\_label = ttk.Label(new\_window, text="Модель:")

model\_label.pack()

model\_combobox = ttk.Combobox(new\_window, values=models)

model\_combobox.pack()

imei\_label = ttk.Label(new\_window, text="IMEI:")

imei\_label.pack()

imei\_combobox = ttk.Combobox(new\_window, values=imeis)

imei\_combobox.pack()

filter\_button = tk.Button(new\_window, text="Применить фильтр", command=lambda: filter\_data(tree, brand\_combobox, model\_combobox, imei\_combobox))

filter\_button.pack(pady=5)

date\_entry = DateEntry(new\_window, width=12, background='darkblue', foreground='white', date\_pattern='yyyy-mm-dd')

date\_entry.pack(pady=5)

update\_button = tk.Button(new\_window, text="Обновить дату покупки", command=lambda: update\_purchase\_date(tree, date\_entry))

update\_button.pack(pady=5)

tree.bind("<Double-1>", lambda event: on\_item\_select(event, tree, date\_entry))

def open\_add\_service\_window():

# Создаем новое окно

window = Toplevel(root)

window.title("Добавление услуги")

window.geometry("220x250")

# Поле для ввода названия услуги

tk.Label(window, text="Название услуги:").pack(pady=5)

service\_name\_entry = tk.Entry(window)

service\_name\_entry.pack(pady=5)

# Поле для ввода цены услуги

tk.Label(window, text="Цена услуги:").pack(pady=5)

service\_price\_entry = tk.Entry(window)

service\_price\_entry.pack(pady=5)

# Выбор даты

tk.Label(window, text="Дата продажи:").pack(pady=5)

sale\_date\_entry = DateEntry(window, width=12, background='darkblue', foreground='white', borderwidth=2, pady=20)

sale\_date\_entry.pack(pady=5)

#sale\_date\_entry.insert(0, datetime.now().strftime('%Y-%m-%d')) # Устанавливаем текущую дату по умолчанию

def save\_service():

service\_name = service\_name\_entry.get()

service\_price = service\_price\_entry.get()

sale\_date = sale\_date\_entry.get()

if service\_name and service\_price and sale\_date:

try:

service\_price = float(service\_price)

connections = sqlite3.connect('my\_database.db')

cursor = connections.cursor()

# Сохраняем запись в базу данных

cursor.execute('INSERT INTO Services (name, price, data\_of\_sale\_services) VALUES (?, ?, ?)',(service\_name, service\_price, sale\_date))

connections.commit()

connection.close()

messagebox.showinfo("Успешно", "Запись успешно добавлена!")

window.destroy() # Закрываем окно после сохранения

except ValueError:

messagebox.showwarning("Ошибка", "Цена должна быть числом!")

else:

messagebox.showwarning("Ошибка", "Пожалуйста, заполните все поля!")

# Кнопка для сохранения услуги

save\_button = tk.Button(window, text="Сохранить", command=save\_service)

save\_button.pack(pady=10)

def open\_table\_window(action):

table\_window = tk.Toplevel()

table\_window.title(f"{action} записи")

table\_window.geometry('1200x400')

table\_label = tk.Label(table\_window, text="Выберите таблицу:")

table\_label.pack()

global table\_combobox

table\_combobox = ttk.Combobox(table\_window, values=["telephone", "Accessories", "Services"])

table\_combobox.pack()

global tree1

tree1 = ttk.Treeview(table\_window, show='headings')

tree1.pack()

def update\_treeview\_columns(table\_name):

# Получаем список столбцов для выбранной таблицы

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

cursor.execute(f"PRAGMA table\_info({table\_name})") # Получаем информацию о таблице

columns = cursor.fetchall()

tree1["columns"] = [col[1] for col in columns] # Устанавливаем столбцы по имени

for col in tree1["columns"]:

tree1.heading(col, text=col) # Устанавливаем заголовки столбцов

# Очищаем Treeview и загружаем данные

for item in tree1.get\_children():

tree1.delete(item)

cursor.execute(f"SELECT \* FROM {table\_name}")

records = cursor.fetchall()

for record in records:

tree1.insert("", "end", values=record)

connection.close()

def load\_data():

table\_name = table\_combobox.get()

if table\_name:

update\_treeview\_columns(table\_name)

load\_button = tk.Button(table\_window, text="Загрузить данные", command=load\_data)

load\_button.pack()

def select\_record():

selected\_item = tree1.selection()

if selected\_item:

values = tree1.item(selected\_item)['values']

edit\_or\_delete\_record(values, action)

select\_button = tk.Button(table\_window, text="Выбрать запись", command=select\_record)

select\_button.pack()

def edit\_or\_delete\_record(values, action):

if action == "Редактировать":

edit\_record\_window = tk.Toplevel()

edit\_record\_window.title("Редактировать запись")

entries = []

for i, value in enumerate(values[1:], start=1): # Пропускаем ID

entry\_label = tk.Label(edit\_record\_window, text=f"Значение для {tree1['columns'][i]}:")

entry\_label.pack()

entry = tk.Entry(edit\_record\_window)

entry.insert(0, value)

entry.pack()

entries.append(entry)

def save\_changes():

new\_values = [entry.get() for entry in entries]

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

if table\_combobox.get() == "telephone":

cursor.execute(f"""

UPDATE telephone

SET name = ?, brand = ?, IMEI = ?, Date\_of\_admission = ?,

Date\_of\_purchase = ?, Admission\_price = ?, Sale\_price = ?, processor = ?, RAM = ?, OS = ?

WHERE id = ?""", (\*new\_values, values[0]))

elif table\_combobox.get() == "Accessories":

cursor.execute(f"""

UPDATE Accessories

SET telephone\_id = ?, name = ?, price = ?, date\_of\_sale = ?

WHERE id = ?""", (\*new\_values, values[0]))

elif table\_combobox.get() == "Services":

cursor.execute(f"""

UPDATE Services

SET name = ?, price = ?, data\_of\_sale\_services = ?

WHERE id = ?""", (\*new\_values, values[0]))

connection.commit()

connection.close()

edit\_record\_window.destroy()

messagebox.showinfo("Успех", "Запись обновлена.")

save\_button = tk.Button(edit\_record\_window, text="Сохранить изменения", command=save\_changes)

save\_button.pack()

elif action == "Удалить":

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

cursor.execute(f"DELETE FROM {table\_combobox.get()} WHERE id = ?", (values[0],))

connection.commit()

connection.close()

messagebox.showinfo("Успех", "Запись удалена.")

def show\_image(image\_path):

if not image\_path:

messagebox.showerror("Ошибка", "Изображение не найдено!")

return

try:

image = Image.open(image\_path)

image.thumbnail((500, 500)) # уменьшаем для отображения

photo = ImageTk.PhotoImage(image)

image\_window = tk.Toplevel()

image\_window.title("Изображение")

image\_label = tk.Label(image\_window, image=photo)

image\_label.image = photo # сохраняем ссылку для избежания garbage collection

image\_label.pack()

except FileNotFoundError:

messagebox.showerror("Ошибка", "Файл изображения не найден.")

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось загрузить изображение: {e}")

def on\_treeview\_single\_click(event, tree):

"""Обработчик одинарного клика для выбора записи."""

# Одинарный клик теперь делает только выделение, для отображения ничего не делаем.

pass

def on\_treeview\_double\_click(event, tree):

"""Обработчик двойного клика для отображения изображения."""

selected\_item = tree.selection()

if not selected\_item:

return

item\_values = tree.item(selected\_item)['values']

if item\_values and len(item\_values) > 0:

# Assuming image\_path is the last column in values

image\_path\_index = len(item\_values) - 1 # index for last item

if image\_path\_index >= 0:

image\_path = item\_values[image\_path\_index]

if image\_path: # Check if the path exists

show\_image(image\_path)

def add\_photo\_to\_record(tree, table\_name, root):

"""Позволяет пользователю выбрать изображение и сохранить его путь в базу данных."""

selected\_item = tree.selection()

if not selected\_item:

messagebox.showerror("Ошибка", "Выберите запись, в которую нужно добавить фото!")

return

item\_values = tree.item(selected\_item)['values']

if not item\_values:

messagebox.showerror("Ошибка", "Не удалось получить данные о записи!")

return

item\_id = item\_values[0] # получаем id записи

file\_path = filedialog.askopenfilename(

title="Выберите изображение",

filetypes=(("Изображения", "\*.jpg \*.jpeg \*.png"), ("Все файлы", "\*.\*"))

)

if file\_path:

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

try:

cursor.execute(f"UPDATE {table\_name} SET image\_path = ? WHERE id = ?", (file\_path, item\_id))

connection.commit()

messagebox.showinfo("Успех", "Изображение добавлено.")

load\_data\_into\_treeview(tree, table\_name)

except Exception as e:

messagebox.showerror("Ошибка", f"Не удалось обновить базу данных: {e}")

finally:

connection.close()

def load\_data\_into\_treeview(tree, table\_name):

"""Загружает данные из базы данных в Treeview."""

try:

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

cursor.execute(f"SELECT \*, image\_path FROM {table\_name}")

rows = cursor.fetchall()

connection.close()

tree.delete(\*tree.get\_children())

for row in rows:

tree.insert('', 'end', values=row)

except Exception as e:

messagebox.showerror("Ошибка", f"Ошибка при загрузке данных: {e}")

def open\_additional\_window(root):

"""Открывает дополнительное окно с Treeview и кнопками."""

additional\_window = tk.Toplevel(root)

additional\_window.title("Данные из таблицы telephone")

additional\_window.geometry("1400x300")

# Создание Treeview

tree = ttk.Treeview(additional\_window)

tree.pack(expand=True, fill="both")

# Загрузка данных в Treeview

table\_name = "telephone"

load\_data\_into\_treeview(tree, table\_name)

#Загружаем заголовки столбцов

tree['columns'] = get\_table\_columns(table\_name)

for col in tree['columns']:

tree.heading(col, text=col)

# Привязка события выбора к функции

tree.bind('<ButtonRelease-1>', lambda event: on\_treeview\_single\_click(event, tree))

tree.bind('<Double-Button-1>', lambda event: on\_treeview\_double\_click(event, tree))

# Кнопка "Добавить фото"

add\_photo\_button = ttk.Button(additional\_window, text="Добавить фото", command=lambda: add\_photo\_to\_record(tree, table\_name, additional\_window))

add\_photo\_button.pack()

def get\_table\_columns(table\_name):

connection = sqlite3.connect('my\_database.db')

cursor = connection.cursor()

cursor.execute(f"PRAGMA table\_info({table\_name})")

columns = [row[1] for row in cursor.fetchall()]

connection.close()

return columns

# Создаем основное окно

root = tk.Tk()

root.title("База данных магазина")

root.geometry("1540x900")

root.resizable(False, False)

treeview\_frame = tk.Frame(root)

treeview\_frame.place(relx=0.5, rely=0.5, anchor=tk.CENTER)

frame = tk.Frame(root, bg='lightblue')

frame.pack(padx=20, pady=20) # Добавляем отступы вокруг Frame

add\_accessory\_button = tk.Button(frame, text="Добавить аксессуар", command=open\_accessory\_window)

add\_accessory\_button.grid(row=1,column=2)

btn4 = tk.Button(frame, text="Прибыль по телефонам", height=1, width=20, command=open\_date\_window)

btn4.grid(row=1,column=1)

add\_service\_btn = tk.Button(frame, text="Оказать услугу", command=open\_add\_service\_window)

add\_service\_btn.grid(row=1, column=3)

combobox = ttk.Combobox(frame, values=table\_names)

combobox.bind("<<ComboboxSelected>>", on\_combobox\_select)

combobox.grid(row=2, column=10, padx=10, pady=10)

btn = tk.Button(frame, text="Добавить телефон", command=open\_add\_window)

btn.grid(row=1, column=4)

delete\_button = tk.Button(frame, text="Удалить запись", command=lambda: open\_table\_window("Удалить"))

delete\_button.grid(row=1, column=8)

edit\_button = tk.Button(frame, text="Редактировать запись", command=lambda: open\_table\_window("Редактировать"))

edit\_button.grid(row=1, column=9)

open\_window\_button = tk.Button(frame, text="Продать телефон", command=open\_new\_window)

open\_window\_button.grid(row=1, column=5)

btn2 = tk.Button(frame, text="Не проданные телефоны", command=show\_sold\_items\_1)

btn2.grid(row=1,column=6)

btn3 = tk.Button(frame, text="Проданные телефоны", command=show\_sold\_items)

btn3.grid(row=1,column=7)

open\_window\_button = tk.Button(frame, text="Открыть окно для просмотра фото", command=lambda: open\_additional\_window(root))

open\_window\_button.grid(row=1, column=10)

# Настройка ширины Frame в зависимости от содержимого

frame.update\_idletasks() # Обновляем размеры Frame после добавления виджетов

frame.config(width=frame.winfo\_width())

root.mainloop()