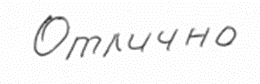
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ

КУРСОВАЯ РАБОТА (ПРОЕКТ)  ЗАЩИЩЕНА С ОЦЕНКОЙ

|  |  |  |
| --- | --- | --- |
| РУКОВОДИТЕЛЬ  Д-р техн. Наук, доцент | подпись, дата | В.А. Галанина |
| должность, уч. степень, звание | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ |
| Проектирование базы данных для учёта домашних финансов |
| по дисциплине: БАЗЫ ДАННЫХ |

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

М112

А.А. Поспелова

, 30.05.2024

подпись, дата инициалы, фамилия

Санкт-Петербург 2024

**СОДЕРЖАНИЕ**

Содержание2

Введение3

1. Построение информационной модели пред. обл.4

1.1. Описание предметной области4

1.2. Технические требования4

2. Построение инфологической модели5

2.1. Определение объектов (сущностей) пред. обл. и их атрибутов5

2.2. Выявление связей между сущностями5

2.3. Построение ER-диаграммы6

3. Формирование реляционной базы данных6

3.1. Разработка сущностей в базе данных6

3.2. Использование триггеров и функций

3.3. Вставка тестовых данных19

4. Разработка тестового приложение21

5. Демонстрация интерфейса38

Заключение52

Список использованных источников53

**ВВЕДЕНИЕ**

Для реализации заданной информационной системы «Домашние финансы» требуется создать визуальное приложение.

В результате работы разработана база данных с использованием языка PostgreSQL, а интерфейс взаимодействия с ней выполнен на языке программирования Python. Программа позволяет легко взаимодействовать с базой данных финансов, вести учёт доходов, расходов, а также позволяет корректно учитывать отложенные накопления.

**ОСНОВНАЯ ЧАСТЬ**

1. **Построение информационной модели предметной области.**
   1. **Описание предметной области.**

База данных для учёта домашних финансов предназначена для отслеживания доходов и расходов всех членов семьи. В базе хранится информация о бюджете, поступлениях и платежах, а также об отложенных средствах. Каждый пользователь приложения имеет возможность оперировать с денежными средствами.

Пользователи делятся на две категории: администратор и обычный пользователь. Администратор в базе может быть только один, у него расширенный функционал, выражающийся в возможности редактировать и удалять записи с пополнениями и списаниями, а также редактировать категории доходов и расходов: добавлять новые, удалять ненужные. При входе в приложение каждый пользователь указывает свои ФИО и личный пароль. Использование базы данных предполагается только взрослыми членами семьи.

Пополнение копилки происходит только через основной баланс. Если пользователь с какого-либо дохода (например, с зарплаты) желает отложить часть денег в копилку, ему сначала необходимо зачислить средства в основной бюджет и только после этого он сможет перевести часть средств в копилку.

Для того, чтобы воспользоваться деньгами копилки, сначала средства переводятся на основной счёт, и только затем можно сформировать новый расход.

* 1. **Технические требования**

Приложение должно выполнять следующие функции:

* Ведение базы данных пользователей;
* Ведение записей об истории доходов и расходов;
* Ведение записей о взаимодействии с «копилкой».

1. **Построение инфологической модели.**
   1. **Определение объектов (сущностей) предметной области и их атрибутов.**

Сущности предметной области:

* Пользователи

Пользователями являются все члены семьи, использующие БД. В базе хранится ФИО, статус и пароль для доступа. Также каждый пользователь имеет уникальный ID.

* Статусы

У каждого пользователя есть статус. Каждый статус имеет ID и название.

* Доходы

Доходом является любое пополнение счёта. При записи нового дохода указывается его значение (сумма пополнения), дата зачисления и категория. Каждое пополнение имеет свой уникальный ID и всегда связывается с ID пользователя, который вошёл в систему.

* Расходы

Расходом является любое списание со счёта. При записи нового расхода указывается его значение (сумма затраты), дата списания и категория. Каждый расход имеет свой уникальный ID и всегда связывается с ID пользователя, который вошёл в систему.

* Копилка

Копилка нужна для откладывания денег на крупные покупки (автомобиль, недвижимость и пр.) У копилки есть свой баланс. Для пополнения копилки указывается значение перевода и дата. У каждого перевода есть свой уникальный ID.

* Категории\_доход

Существуют различные типы доходов. Каждый тип имеет свой ID и название.

* Категории\_расход

Существуют различные типы расходов. Каждый тип имеет свой ID и название.

* 1. **Выявление связей между сущностями**
* Связь «Пользователи» и «Доходы» (один ко многим):

Один пользователь может создать несколько записей о доходах.

* Связь «Пользователи» и «Расходы» (один ко многим):

Один пользователь может создать несколько записей о расходах.

* Связь «Пользователи» и «Копилка» (один ко многим):

Один пользователь может создать несколько записей о переводах.

* Связь «Статусы» и «Пользователи» (один ко многим):

У нескольких пользователей может быть один статус.

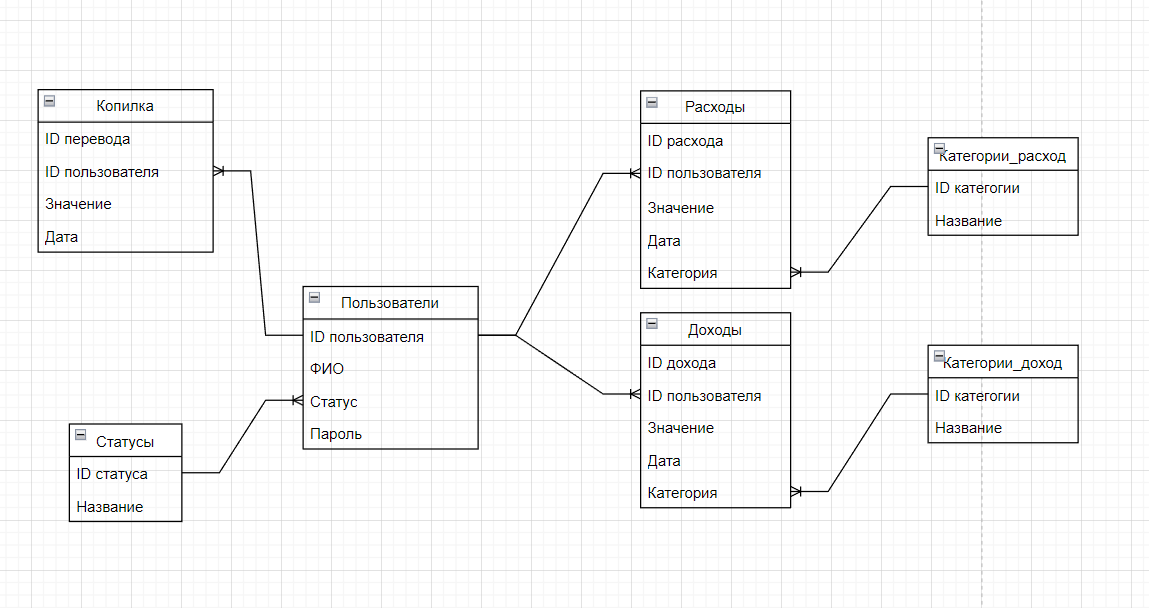
* Связь «Категории\_доход» и «Доходы» (один ко многим):

Несколько записей о доходах может иметь одну категорию.

* Связь «Категории\_расход» и «Расходы» (один ко многим):

Несколько записей о доходах может иметь одну категорию

* 1. **Построение ER-диаграммы.**



*Рисунок 1- Инфологическая модель (ER-диаграмма)*

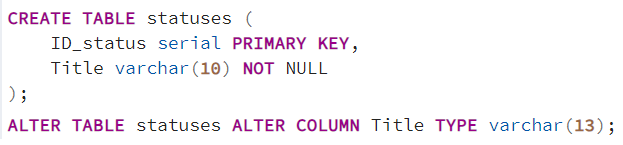
1. **Формирование реляционной базы данных.**
   1. **Разработка сущностей в базе данных**

Разработка сущностей базы данных является ключевым этапом в проектировании информационной системы, так как она обеспечивает структуру данных и отношения между ними. В данном проекте используется PostgreSQL для хранения данных, и сущности спроектированы таким образом, чтобы обеспечить надежность, целостность и производительность базы данных.



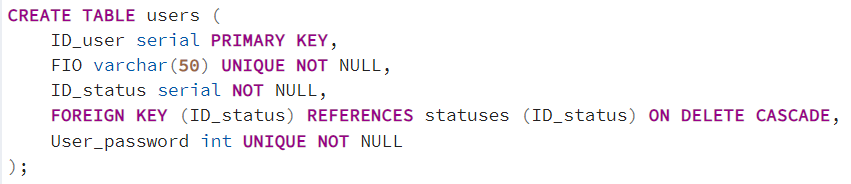
*Рисунок 2 – Создание схемы для базы данных*

Рассмотрим таблицы и их взаимосвязи.



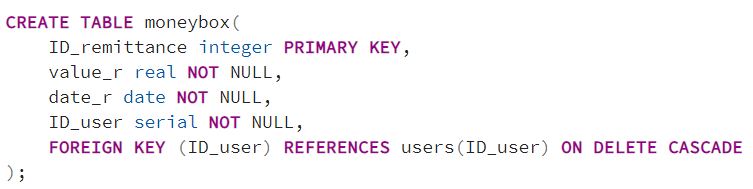
*Рисунок 3 – Создание таблицы «Статусы»*

Таблица «Статусы» (statuses) содержит два поля: ID статуса и название. ID статуса (ID\_status) является ключевым и уникальным полем и нужен для связи с другими таблицами. Поле «title» содержит название каждого статуса. Данная таблица имеет всего две записи, поскольку использование данного приложения предполагается несколькими обычными пользователями и одним администратором.



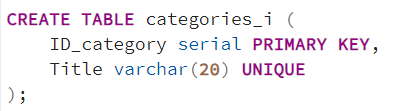
*Рисунок 4 – Создание таблицы «Пользователи»*

Таблица «Пользователи» (users) содержит четыре поля: ID пользователя, ФИО, ID статуса и пароль. В таблице хранится информация о всех пользователях. У каждого клиента приложения есть свой уникальный ID (ID\_user), он нужен для связи с другими таблицами. Поля ФИО (FIO) и пароль (User\_password) нужны для авторизации пользователя в приложении, каждое поле должно быть уникально. ID статуса (ID\_status) взято из таблицы «statuses».



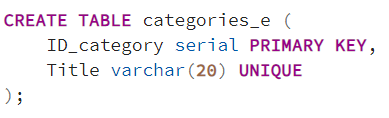
*Рисунок 5 – Создание таблицы «Копилка»*

Таблица «Копилка» (moneybox) содержит четыре поля: ID перевода, значение, дата и ID пользователя. Таблица содержит информацию об операциях перевода средств из основного бюджета в копилку и наоборот. У каждого перевода есть свой уникальный ID (ID\_remittance), он нужен для работы с записями в таблице. Поле значение (value) – размер денежных средств для перевода, поле дата (date) – дата перевода. ID пользователя (ID\_user) взято из таблицы «users» - обозначает пользователя, выполнившего перевод.



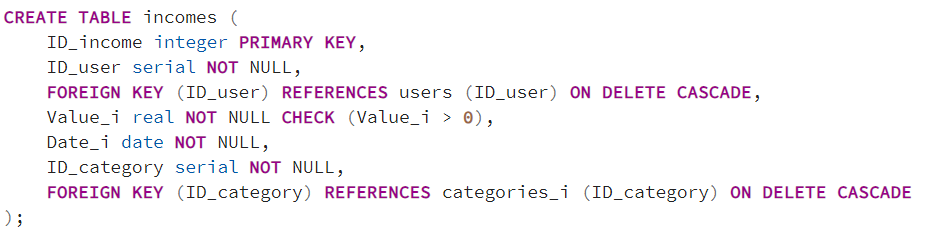
*Рисунок 6 – Создание таблицы «Категории\_доход»*

Таблица «Категории\_доход» (categories\_i) содержит два поля: ID категории и название. Таблица содержит информацию о возможных типах прибыли, которые есть у пользователей. Каждая категория имеет свой уникальный ID (ID\_category), он нужен для связи с другими таблицами. Поле с названием (title) содержит название типа прибыли и является уникальным. Таким образом, формируется список доходов.



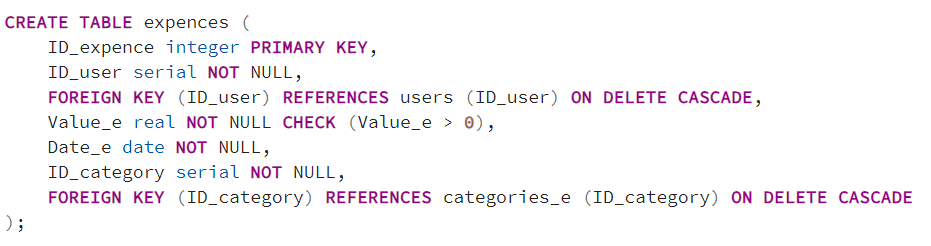
*Рисунок 7 – Создание таблицы «Категории\_расход»*

Таблица «Категории\_расход» (categories\_e) содержит два поля: ID категории и название. Таблица содержит информацию о возможных типах затрат, которые есть у пользователей. Каждая категория имеет свой уникальный ID (ID\_category), он нужен для связи с другими таблицами. Поле с названием (title) содержит название типа затрат и является уникальным. Таким образом, формируется список расходов.



*Рисунок 8 – Создание таблицы «Доходы»*

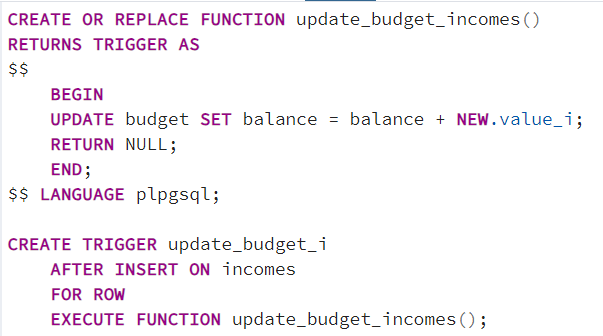
Таблица «Доходы» (incomes) содержит пять полей: ID дохода, ID пользователя, значение, дата и ID категории. Таблица содержит информацию об операциях пополнения основного бюджета. У каждого пополнения есть свой уникальный ID (ID\_income), он нужен для работы с записями в таблице. Поле значение (value\_i) – размер денежных средств для пополнения – должно быть больше нуля, поле дата (date\_i) – дата пополнения. ID категории (ID\_category) взято из таблицы «categories\_i» – обозначает тип полученной прибыли. ID пользователя (ID\_user) взято из таблицы «users» – обозначает пользователя, выполнившего зачисление.



*Рисунок 9 – Создание таблицы «Расходы»*

Таблица «Расходы» (expences) содержит пять полей: ID расхода, ID пользователя, значение, дата и ID категории. Таблица содержит информацию об операциях списания средств их основного бюджета. У каждого списания есть свой уникальный ID (ID\_expence), он нужен для работы с записями в таблице. Поле значение (value\_e) – размер денежных средств для списания – должно быть больше нуля, поле дата (date\_e) – дата списания. ID категории (ID\_category) взято из таблицы «categories\_e» – обозначает тип затрат. ID пользователя (ID\_user) взято из таблицы «users» – обозначает пользователя, выполнившего списание.

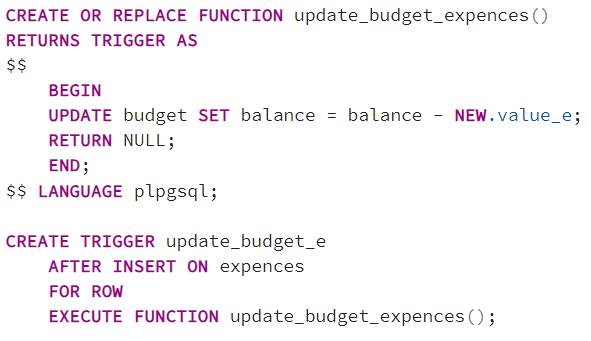
* 1. Использование триггеров и функций



*Рисунок 10 – Триггер и функция для обновления бюджета*

Функция update\_budget\_incomes() обновляет значение баланса, добавляя к уже имеющимся средствам значение нового зачисления средств.

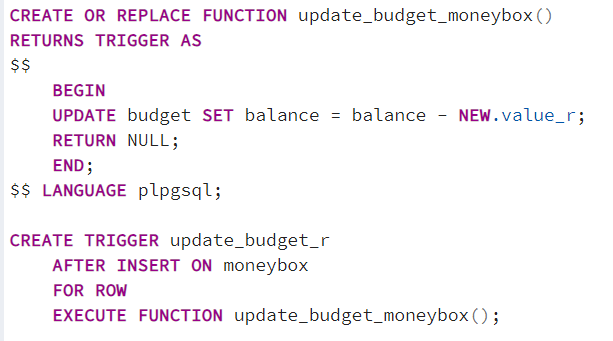
Триггер update\_budget\_i срабатывает после создания новой записи в таблице «Доходы» (incomes), вызывая функцию update\_budget\_incomes().



*Рисунок 11 – Триггер и функция для обновления бюджета*

Функция update\_budget\_expences() обновляет значение баланса, вычитая из имеющихся средств значение нового списания средств.

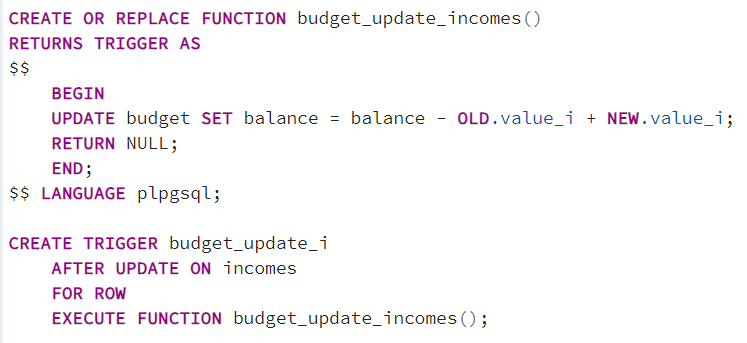
Триггер update\_budget\_e срабатывает после создания новой записи в таблице «Расходы» (expences), вызывая функцию update\_budget\_ expences().



*Рисунок 12 – Триггер и функция для обновления бюджета*

Функция update\_budget\_moneybox() обновляет значение баланса, вычитая из имеющихся средств значение нового перевода средств в копилку.

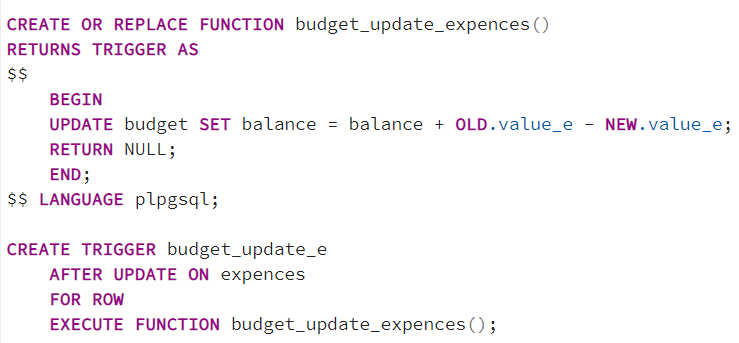
Триггер update\_budget\_r срабатывает после создания новой записи в таблице «Копилка» (moneybox), вызывая функцию update\_budget\_ moneybox().



*Рисунок 13 – Триггер и функция для обновления бюджета*

Функция budget\_update\_incomes() обновляет значение баланса, вычитая из имеющихся средств старое значение зачисления и прибавляя новое значение зачисления средств в бюджете.

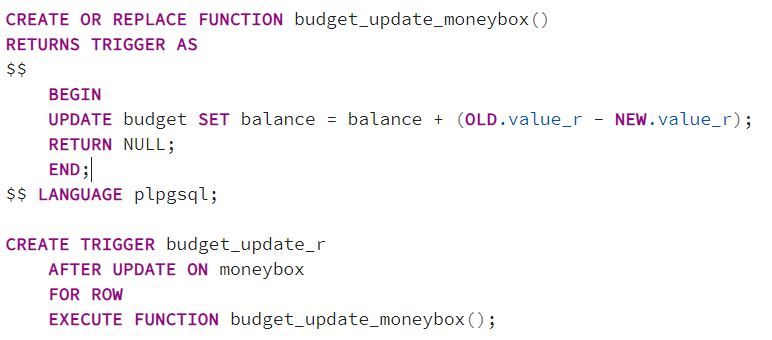
Триггер budget\_update\_i срабатывает после изменения записи в таблице «Доходы» (incomes), вызывая функцию budget\_update\_incomes(). В результате получается, что, если запись в таблице «Доходы» была изменена и значение зачисления тоже подверглось изменению, то значение основного бюджета изменилось на разницу старого и нового значений зачисления.



*Рисунок 14 – Триггер и функция для обновления бюджета*

Функция budget\_update\_expences() обновляет значение баланса, прибавляя к имеющимся средствам старое значение расхода и вычитая новое значение расхода средств в бюджете.

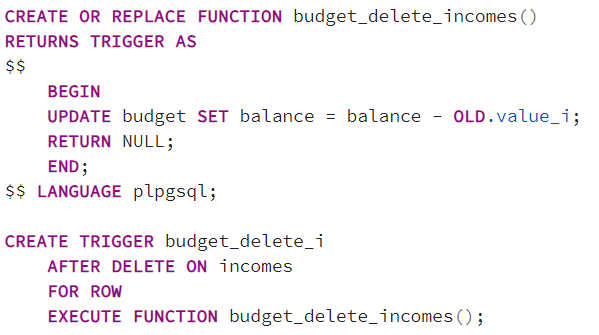
Триггер budget\_update\_e срабатывает после изменения записи в таблице «Расходы» (expences), вызывая функцию budget\_update\_expences(). В результате получается, что, если запись в таблице «Расходы» была изменена и значение списания тоже подверглось изменению, то значение основного бюджета изменилось на разницу нового и старого значений расхода.



*Рисунок 15 – Триггер и функция для обновления бюджета*

Функция budget\_update\_moneybox() обновляет значение баланса, прибавляя к имеющимся средствам старое значение перевода и вычитая новое значение перевода средств из копилки в бюджете.

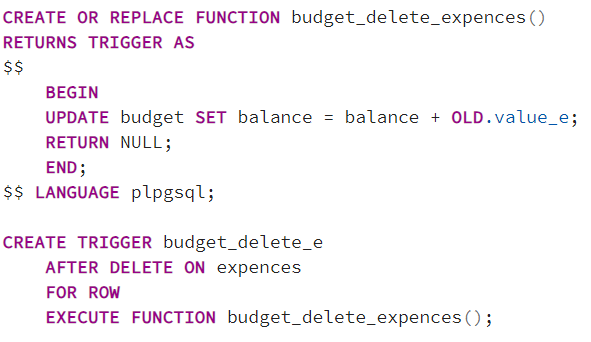
Триггер budget\_update\_r срабатывает после изменения записи в таблице «Копилка» (moneybox), вызывая функцию budget\_update\_moneybox(). В результате получается, что, если запись в таблице «Копилка» была изменена и значение перевода тоже подверглось изменению, то значение основного бюджета изменилось на разницу нового и старого значений перевода.



*Рисунок 15 – Триггер и функция для обновления бюджета*

Функция budget\_delete\_incomes() обновляет значение баланса, вычитая из имеющихся средств старое значение зачисления.

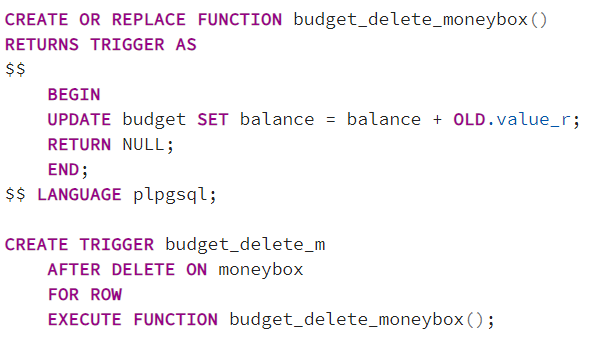
Триггер budget\_delete\_i срабатывает после удаления записи из таблицы «Доходы» (incomes), вызывая функцию budget\_delete\_incomes(). В результате получается, что, если запись в таблице «Доходы» была удалена, то значение основного бюджета сократилось на старое значение зачисления.



*Рисунок 16 – Триггер и функция для обновления бюджета*

Функция budget\_delete\_expences() обновляет значение баланса, прибавляя к имеющимся средствам старое значение списания.

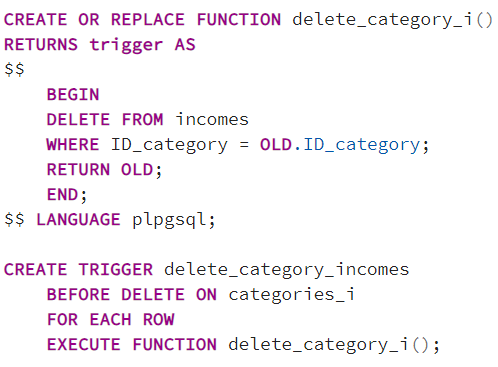
Триггер budget\_delete\_e срабатывает после удаления записи из таблицы «Расходы» (expences), вызывая функцию budget\_delete\_expences(). В результате получается, что, если запись в таблице «Расходы» была удалена, то значение основного бюджета увеличилось на старое значение списания.



*Рисунок 17 – Триггер и функция для обновления бюджета*

Функция budget\_delete\_moneybox() обновляет значение баланса, прибавляя к имеющимся средствам старое значение перевода.

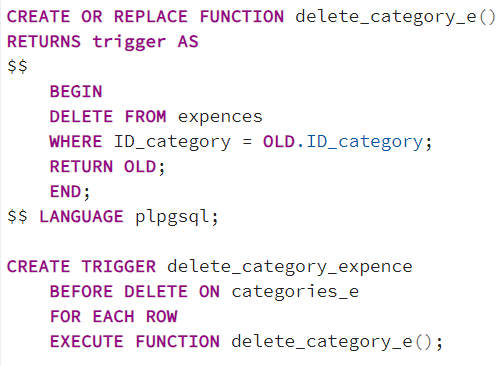
Триггер budget\_delete\_r срабатывает после удаления записи из таблицы «Копилка» (moneybox), вызывая функцию budget\_delete\_moneybox(). В результате получается, что, если запись в таблице «Копилка» была удалена, то значение основного бюджета увеличилось на старое значение перевода.



*Рисунок 18 – Триггер и функция для удаления категории*

Функция delete\_category\_i() удаляет записи о доходе из таблицы «Доходы» (incomes), где ID категории соответствует передаваемому ID категории.

Триггер delete\_category\_incomes срабатывает перед удалением записи из таблицы «Категории\_доход» (categories\_i), вызывая функцию delete\_category\_i (). В результате получается, что, если запись в таблице «Категории\_доход» была удалена, то все записи, относящиеся к удалённой категории, будут удалены из таблицы «Доходы».



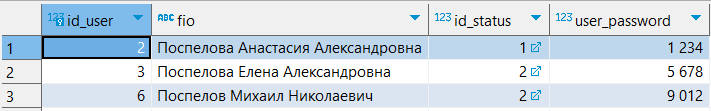
*Рисунок 19 – Триггер и функция для удаления категории*

Функция delete\_category\_e() удаляет записи о доходе из таблицы «Расходы» (expences), где ID категории соответствует передаваемому ID категории.

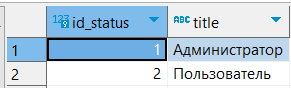
Триггер delete\_category\_expence срабатывает перед удалением записи из таблицы «Категории\_расход (categories\_e), вызывая функцию delete\_category\_e (). В результате получается, что, если запись в таблице «Категории\_расход» была удалена, то все записи, относящиеся к удалённой категории, будут удалены из таблицы «Доходы».

* 1. **Вставка тестовых данных**

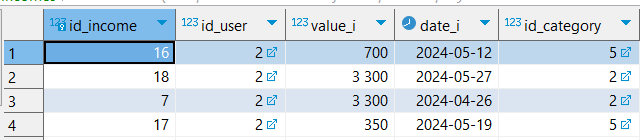
Для обеспечения работоспособности системы и возможности тестирования были вставлены начальные данные во все таблицы.



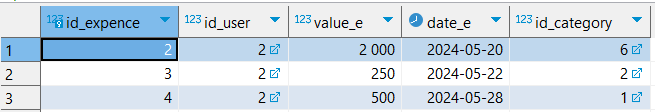
*Рисунок 20 – Тестовые данные в таблице «users»*



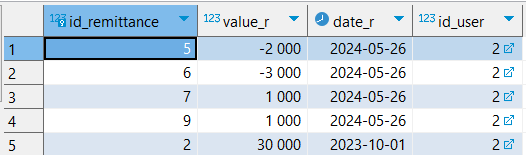
*Рисунок 21 – Тестовые данные в таблице «statuses»*



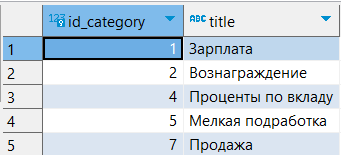
*Рисунок 22 – Тестовые данные в таблице «incomes»*



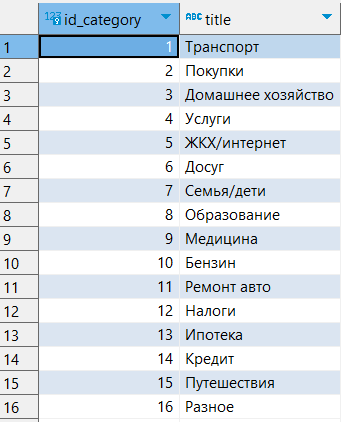
*Рисунок 23 – Тестовые данные в таблице «expences»*



*Рисунок 24 – Тестовые данные в таблице «moneybox»*



*Рисунок 25 – Тестовые данные в таблице «categories\_i»*



*Рисунок 26 – Тестовые данные в таблице «categories\_e»*

1. **Разработка тестового приложения.**

Для разработки приложения были использованы две основные библиотеки: «tkinter» и «psycopg2». Библиотека «psycopg2» используется для работы с базой данных PostgreSQL, а «tkinter» для разработки графического интерфейса.

Серверная часть: «*database.py*»

import psycopg2  
  
connection = psycopg2.connect(user="postgres",  
 password="Fuckingpassword1",  
 host="127.0.0.1",  
 port="5432",  
 database="kursach")  
cursor = connection.cursor()  
cursor.execute('SET search\_path TO home\_finance,public')  
  
  
def check\_admin\_unique(status):  
 if status == 2:  
 return True  
 else:  
 sql = 'SELECT ID\_status FROM users'  
 cursor.execute(sql)  
 users\_statuses = tuple(i[0] for i in cursor.fetchall())  
 if status not in users\_statuses:  
 return True  
 else:  
 return False  
  
  
def add\_user\_d(new\_name, new\_password, status):  
 sql = 'INSERT INTO users (FIO, ID\_status, user\_password)'\  
 f'VALUES (\'{new\_name}\', {status}, {new\_password})'  
 cursor.execute(sql)  
 connection.commit()  
  
  
def check\_user\_password(user\_FIO):  
 sql = f'SELECT user\_password FROM users WHERE FIO = \'{user\_FIO}\''  
 cursor.execute(sql)  
 return cursor.fetchone()[0]  
  
  
def get\_balance():  
 sql = 'SELECT balance FROM budget'  
 cursor.execute(sql)  
 return cursor.fetchone()[0]  
  
  
def get\_balance\_r():  
 sql = 'SELECT SUM(value\_r) FROM moneybox'  
 cursor.execute(sql)  
 return cursor.fetchone()[0]  
  
  
def get\_users():  
 sql = 'SELECT FIO FROM users'  
 cursor.execute(sql)  
 users = tuple(i[0] for i in cursor.fetchall())  
 return users  
  
  
def get\_user\_status(userID):  
 sql = 'SELECT ID\_status FROM users ' \  
 f'WHERE ID\_user = {userID}'  
 cursor.execute(sql)  
 return int(cursor.fetchone()[0])  
  
  
def get\_categories\_i():  
 sql = 'SELECT title FROM categories\_i'  
 cursor.execute(sql)  
 categories\_i = tuple(i[0] for i in cursor.fetchall())  
 return categories\_i  
  
  
def get\_categories\_e():  
 sql = 'SELECT title FROM categories\_e'  
 cursor.execute(sql)  
 categories\_e = tuple(i[0] for i in cursor.fetchall())  
 return categories\_e  
  
  
def delete\_category\_i\_e(title, flag):  
 if not flag:  
 sql = f"DELETE FROM categories\_i WHERE id\_category = (" \  
 f"SELECT id\_category WHERE title = '{title}')"  
 else:  
 sql = f"DELETE FROM categories\_e WHERE id\_category = (" \  
 f"SELECT id\_category WHERE title = '{title}')"  
 cursor.execute(sql)  
 connection.commit()  
  
  
def get\_userID(userFIO):  
 sql = f'SELECT ID\_user FROM users WHERE FIO = \'{userFIO}\''  
 cursor.execute(sql)  
 return int(cursor.fetchone()[0])  
  
  
def get\_incomes():  
 sql = 'SELECT ID\_income, value\_i, date\_i, title, FIO FROM incomes ' \  
 'INNER JOIN categories\_i USING(ID\_category) ' \  
 'INNER JOIN users USING(ID\_user)'  
 cursor.execute(sql)  
 return cursor.fetchall()  
  
  
def get\_expenses():  
 sql = 'SELECT ID\_expence, value\_e, date\_e, title, FIO FROM expences ' \  
 'INNER JOIN categories\_e USING(ID\_category) ' \  
 'INNER JOIN users USING(ID\_user)'  
 cursor.execute(sql)  
 return cursor.fetchall()  
  
  
def get\_remittance():  
 sql = 'SELECT ID\_remittance, value\_r, date\_r, FIO FROM moneybox ' \  
 'INNER JOIN users USING(ID\_user)'  
 cursor.execute(sql)  
 return cursor.fetchall()  
  
  
def get\_ID(flag):  
 if not flag:  
 sql = 'SELECT ID\_income FROM incomes'  
 else:  
 sql = 'SELECT ID\_expence FROM expences'  
 cursor.execute(sql)  
 return cursor.fetchall()  
  
  
def get\_ID\_r():  
 sql = 'SELECT ID\_remittance FROM moneybox'  
 cursor.execute(sql)  
 return cursor.fetchall()  
  
  
def add\_new\_category(title, flag):  
 if not flag:  
 sql = f"INSERT INTO categories\_i (title) VALUES ('{title}')"  
 else:  
 sql = f"INSERT INTO categories\_e (title) VALUES ('{title}')"  
 cursor.execute(sql)  
 connection.commit()  
  
  
def add\_income\_values(ID\_user, value\_i, date\_i, title):  
 sql = f'SELECT ID\_category FROM categories\_i WHERE title = \'{title}\''  
 cursor.execute(sql)  
 ID\_category = int(cursor.fetchone()[0])  
 sql = 'INSERT INTO incomes(ID\_user, value\_i, date\_i, ID\_category) ' \  
 f'VALUES ({ID\_user}, {value\_i}, \'{date\_i}\', {ID\_category})'  
 cursor.execute(sql)  
 connection.commit()  
  
  
def add\_expense\_values(ID\_user, value\_e, date\_e, title):  
 sql = f'SELECT ID\_category FROM categories\_e WHERE title = \'{title}\''  
 cursor.execute(sql)  
 ID\_category = int(cursor.fetchone()[0])  
 sql = 'INSERT INTO expences(ID\_user, value\_e, date\_e, ID\_category) ' \  
 f'VALUES ({ID\_user}, {value\_e}, \'{date\_e}\', {ID\_category})'  
 cursor.execute(sql)  
 connection.commit()  
  
  
def add\_moneybox\_values(ID\_user, value\_r, date\_r, flag):  
 if flag:  
 sql = 'INSERT INTO moneybox(ID\_user, value\_r, date\_r) '\  
 f'VALUES ({ID\_user}, {value\_r}, \'{date\_r}\')'  
 else:  
 sql = 'INSERT INTO moneybox(ID\_user, value\_r, date\_r) ' \  
 f'VALUES ({ID\_user}, {-int(value\_r)}, \'{date\_r}\')'  
 cursor.execute(sql)  
 connection.commit()  
  
  
def update\_i\_e(id, value, date, category, flag):  
 if not flag:  
 sql = f'SELECT ID\_category FROM categories\_i WHERE title = \'{category}\''  
 cursor.execute(sql)  
 ID\_category = int(cursor.fetchone()[0])  
 sql = f"UPDATE incomes SET (value\_i, date\_i, ID\_category) = ({value}, '{date}', '{ID\_category}') " \  
 f"WHERE ID\_income = {id}"  
 else:  
 sql = f'SELECT ID\_category FROM categories\_e WHERE title = \'{category}\''  
 cursor.execute(sql)  
 ID\_category = int(cursor.fetchone()[0])  
 sql = f"UPDATE expences SET (value\_e, date\_e, ID\_category) = ({value}, '{date}', '{ID\_category}') " \  
 f"WHERE ID\_expence = {id}"  
 cursor.execute(sql)  
 connection.commit()  
  
  
def update\_r(id, value, date):  
 sql = f"UPDATE moneybox SET (value\_r, date\_r) = ({value}, '{date}') " \  
 f"WHERE ID\_remittance = {id}"  
 cursor.execute(sql)  
 connection.commit()  
  
  
def delete\_i\_e\_r(id, flag):  
 if not flag:  
 sql = f"DELETE FROM incomes WHERE ID\_income = {id}"  
 elif flag == 1:  
 sql = f"DELETE FROM expences WHERE ID\_expence = {id}"  
 else:  
 sql = f"DELETE FROM moneybox WHERE ID\_remittance = {id}"  
 cursor.execute(sql)  
 connection.commit()

Описание кода:

* Импорт библиотеки «psycobg2» для взаимодействия с БД;
* Подключение к базе данных «kursach» для доступа к схеме данных «home\_finance»;
* Функция «check\_admin\_unique»

Используется при создании нового пользователя для проверки статуса (проверка количества администраторов);

* Функция «add\_user\_d»

Создаёт нового пользователя в базе;

* Функция «check\_user\_password»

Проверяет достоверность пароля;

* Функция «get\_balance»

Возвращает текущий баланс;

* Функция «get\_balance\_r»

Возвращает текущий баланс копилки;

* Функция «get\_users»

Возвращает всех пользователей;

* Функция «get\_user\_status»

Находит статус пользователя по его ID;

* Функция «get\_categories\_i»

Возвращает все категории доходов;

* Функция «get\_categories\_e»

Возвращает все категории расходов;

* Функция «delete\_category\_i\_e»

Удаляет категорию по её названию;

* Функция «get\_user\_ID»

Находит ID пользователя по его ФИО;

* Функция «get\_incomes»

Возвращает все записи о доходах;

* Функция «get\_expenses»

Возвращает все записи о расходах;

* Функция «get\_remittance»

Возвращает все записи о переводах в копилку;

* Функция «get\_ID»

Возвращает все ID записей о доходах/расходах;

* Функция «get\_ID\_r»

Возвращает все ID записей о переводах;

* Функция «add\_new\_category»

Создаёт новую категорию в базе;

* Функция «add\_income\_values»

Создаёт новую запись о доходе;

* Функция «add\_expense\_values»

Создаёт новую запись о расходе;

* Функция «add\_moneybox\_values»

Создаёт новую запись о переводе в копилку;

* Функция «update\_i\_e»

Изменяет запись в таблице incomes/expences;

* Функция «update\_r»

Изменяет запись в таблице moneybox;

* Функция «delete\_i\_e\_r

Удаляет запись из таблицы incomes/expences/moneybox.

Визуальная часть: «*main.py*»

from tkinter import \*  
from tkinter.ttk import Combobox  
from tkinter import ttk  
from tkinter import messagebox  
from databse import \*  
import re  
  
  
def add\_user():  
 new\_name = new\_name\_textbox.get()  
 new\_password = new\_password\_textbox.get()  
 if status\_combo.get() == 'Администратор':  
 new\_status = 1  
 else:  
 new\_status = 2  
 if all(i.isdigit() for i in new\_password) and new\_password != '':  
 if all(i.isalpha() or i == ' ' for i in new\_name) and new\_name != '':  
 if check\_admin\_unique(new\_status):  
 add\_user\_d(new\_name, new\_password, new\_status)  
 messagebox.showinfo('Добавление', 'Новый пользователь добавлен')  
 else:  
 messagebox.showerror('Ошибка', 'Всего может быть не более одного администратора.')  
 else:  
 messagebox.showerror('Оишбка', 'Имя пользователя должно содержать только буквы и не может быть пустым.')  
 else:  
 messagebox.showerror('Оишбка', 'Пароль должен содержать только цифры и не может быть пустым.')  
  
  
def check\_enter():  
 user\_name = name\_combo.get()  
 entered\_password = password\_textbox.get()  
 if user\_name != '' and entered\_password != '':  
 user\_password = check\_user\_password(user\_name)  
 if str(user\_password) == entered\_password:  
 main\_window\_user(get\_userID(user\_name))  
 else:  
 messagebox.showerror('Оишбка', 'Имя пользователя и пароль не совпадают.')  
 else:  
 messagebox.showerror('Оишбка', 'Имя пользователя и пароль не могут быть пустыми.')  
  
  
def show\_table(tab, flag):  
 table = ttk.Treeview(tab)  
 table['columns'] = ('id', 'value', 'date', 'category', 'user')  
 table.column("#0", width=0, stretch=NO)  
 table.column("id", anchor=CENTER, width=30)  
 table.column("value", anchor=CENTER, width=90)  
 table.column("date", anchor=CENTER, width=100)  
 table.column("category", anchor=CENTER, width=180)  
 table.column("user", anchor=CENTER, width=230)  
 table.heading("#0", text="", anchor=CENTER)  
 table.heading("id", text="ID", anchor=CENTER)  
 table.heading("value", text="Значение", anchor=CENTER)  
 table.heading("date", text="Дата", anchor=CENTER)  
 table.heading("category", text="Категория", anchor=CENTER)  
 table.heading("user", text="Пользователь", anchor=CENTER)  
 if flag:  
 data = get\_incomes()  
 else:  
 data = get\_expenses()  
 for i in range(len(data)):  
 table.insert(parent='', index='end', iid=i, text='', values=data[i])  
 return table  
  
  
def show\_table\_moneybox(tab):  
 table = ttk.Treeview(tab)  
 table['columns'] = ('id', 'value', 'date', 'user')  
 table.column("#0", width=0, stretch=NO)  
 table.column("id", anchor=CENTER, width=30)  
 table.column("value", anchor=CENTER, width=80)  
 table.column("date", anchor=CENTER, width=90)  
 table.column("user", anchor=CENTER, width=220)  
 table.heading("#0", text="", anchor=CENTER)  
 table.heading("id", text="ID", anchor=CENTER)  
 table.heading("value", text="Значение", anchor=CENTER)  
 table.heading("date", text="Дата", anchor=CENTER)  
 table.heading("user", text="Пользователь", anchor=CENTER)  
 data = get\_remittance()  
 for i in range(len(data)):  
 table.insert(parent='', index='end', iid=i, text='', values=data[i])  
 return table  
  
  
def add\_income(userID):  
 income\_window = Tk()  
 income\_window.title('Добавить запись')  
 income\_window.geometry('400x325')  
 income\_window.resizable(False, False)  
 income\_window.focus\_set()  
 income\_window.grab\_set()  
  
 value\_label = Label(income\_window, text='Значение', font=('Arial Bold', 15))  
 value\_label.place(x=30, y=50)  
 value\_textbox = Entry(income\_window, width=23)  
 value\_textbox.place(x=200, y=50)  
  
 date\_label = Label(income\_window, text='Дата', font=('Arial Bold', 15))  
 date\_label.place(x=30, y=120)  
 date\_textbox = Entry(income\_window, width=23)  
 date\_textbox.place(x=200, y=120)  
  
 category\_label = Label(income\_window, text='Категория', font=('Arial Bold', 15))  
 category\_label.place(x=30, y=190)  
 category\_combo = Combobox(income\_window)  
 category\_combo['values'] = get\_categories\_i()  
 category\_combo.place(x=200, y=190)  
  
 def insert\_income():  
 value = value\_textbox.get()  
 date = date\_textbox.get()  
 category = category\_combo.get()  
 if value != '' and date != '' and category != '':  
 if all(i.isdigit() for i in value):  
 date\_regex = re.compile(r'^\d{4}-\d{2}-\d{2}$')  
 if date\_regex.match(date):  
 add\_income\_values(userID, value, date, category)  
 messagebox.showinfo('Добавление', 'Новая запись добавлена')  
 income\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Неверный формат даты. (Правильный: "yyyy-mm-dd")')  
 else:  
 messagebox.showerror('Ошибка', 'Введено неверное значение.')  
 else:  
 messagebox.showerror('Ошибка', 'Все поля должны быть заполнены (значение, дата, категория).')  
  
 add\_button = Button(income\_window, text="Пополнить", height=3, width=20, command=insert\_income)  
 add\_button.place(x=110, y=250)  
  
 income\_window.mainloop()  
  
  
def add\_expense(userID):  
 expense\_window = Tk()  
 expense\_window.title('Добавить запись')  
 expense\_window.geometry('400x325')  
 expense\_window.resizable(False, False)  
 expense\_window.focus\_set()  
 expense\_window.grab\_set()  
  
 value\_label = Label(expense\_window, text='Значение', font=('Arial Bold', 15))  
 value\_label.place(x=30, y=50)  
 value\_textbox = Entry(expense\_window, width=23)  
 value\_textbox.place(x=200, y=50)  
  
 date\_label = Label(expense\_window, text='Дата', font=('Arial Bold', 15))  
 date\_label.place(x=30, y=120)  
 date\_textbox = Entry(expense\_window, width=23)  
 date\_textbox.place(x=200, y=120)  
  
 category\_label = Label(expense\_window, text='Категория', font=('Arial Bold', 15))  
 category\_label.place(x=30, y=190)  
 category\_combo = Combobox(expense\_window)  
 category\_combo['values'] = get\_categories\_e()  
 category\_combo.place(x=200, y=190)  
  
 def insert\_expense():  
 value = value\_textbox.get()  
 date = date\_textbox.get()  
 category = category\_combo.get()  
 if value != '' and date != '' and category != '':  
 if all(i.isdigit() for i in value):  
 date\_regex = re.compile(r'^\d{4}-\d{2}-\d{2}$')  
 if date\_regex.match(date):  
 add\_expense\_values(userID, value, date, category)  
 messagebox.showinfo('Добавление', 'Новая запись добавлена')  
 expense\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Неверный формат даты. (Правильный: "yyyy-mm-dd")')  
 else:  
 messagebox.showerror('Ошибка', 'Введено неверное значение.')  
 else:  
 messagebox.showerror('Ошибка', 'Все поля должны быть заполнены (значение, дата, категория).')  
  
 add\_button = Button(expense\_window, text="Заплатить", height=3, width=20, command=insert\_expense)  
 add\_button.place(x=110, y=250)  
  
 expense\_window.mainloop()  
  
  
def add\_moneybox(userID, flag):  
 moneybox\_window = Tk()  
 moneybox\_window.title('Добавить запись')  
 moneybox\_window.geometry('400x325')  
 moneybox\_window.resizable(False, False)  
 moneybox\_window.focus\_set()  
  
 value\_label = Label(moneybox\_window, text='Значение', font=('Arial Bold', 15))  
 value\_label.place(x=30, y=50)  
 value\_textbox = Entry(moneybox\_window, width=23)  
 value\_textbox.place(x=200, y=50)  
  
 date\_label = Label(moneybox\_window, text='Дата', font=('Arial Bold', 15))  
 date\_label.place(x=30, y=120)  
 date\_textbox = Entry(moneybox\_window, width=23)  
 date\_textbox.place(x=200, y=120)  
  
 def insert\_remittance():  
 value = value\_textbox.get()  
 date = date\_textbox.get()  
 if value != '' and date != '':  
 if all(i.isdigit() for i in value):  
 date\_regex = re.compile(r'^\d{4}-\d{2}-\d{2}$')  
 if date\_regex.match(date):  
 add\_moneybox\_values(userID, value, date, flag)  
 messagebox.showinfo('Добавление', 'Новая запись добавлена')  
 moneybox\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Неверный формат даты. (Правильный: "yyyy-mm-dd")')  
 else:  
 messagebox.showerror('Ошибка', 'Введено неверное значение.')  
 else:  
 messagebox.showerror('Ошибка', 'Все поля должны быть заполнены (значение, дата, категория).')  
  
 if flag:  
 add\_button = Button(moneybox\_window, text="Внести", height=3, width=20, command=insert\_remittance)  
 add\_button.place(x=110, y=250)  
 else:  
 add\_button = Button(moneybox\_window, text="Снять", height=3, width=20, command=insert\_remittance)  
 add\_button.place(x=110, y=250)  
  
 moneybox\_window.mainloop()  
  
  
def change\_category(flag):  
 new\_category\_window = Tk()  
 new\_category\_window.title('Новая категория')  
 new\_category\_window.geometry('400x200')  
 new\_category\_window.resizable(False, False)  
 new\_category\_window.focus\_set()  
  
 title\_label = Label(new\_category\_window, text='Название', font=('Arial Bold', 15))  
 title\_label.place(x=50, y=50)  
 title\_entry = Entry(new\_category\_window, width=15)  
 title\_entry.place(x=200, y=55)  
  
 def add\_category():  
 title = title\_entry.get()  
 if title != '':  
 add\_new\_category(title, flag)  
 messagebox.showinfo('Добавление', f'Категория "{title}" добавлена.')  
 new\_category\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Название категории не указано.')  
  
 add\_button = Button(new\_category\_window, text="Добавить", height=2, width=15, command=add\_category)  
 add\_button.place(x=50, y=130)  
  
 def delete\_category():  
 title = title\_entry.get()  
 if title != '':  
 if (title in get\_categories\_i() and flag is False) or \  
 (title in get\_categories\_e() and flag is True):  
 delete\_category\_i\_e(title, flag)  
 messagebox.showinfo('Удаление', f'Категория "{title}" удалена.')  
 new\_category\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Категории с таким названием не найдено.')  
 else:  
 messagebox.showerror('Ошибка', 'Название категории не указано.')  
  
 delete\_button = Button(new\_category\_window, text="Удалить", height=2, width=15, command=delete\_category)  
 delete\_button.place(x=220, y=130)  
  
 new\_category\_window.mainloop()  
  
  
def update\_operation(flag):  
 update\_window = Tk()  
 update\_window.title('Изменить запись')  
 update\_window.geometry('400x325')  
 update\_window.resizable(False, False)  
 update\_window.focus\_set()  
 update\_window.grab\_set()  
  
 id\_label = Label(update\_window, text='ID', font=('Arial Bold', 10))  
 id\_label.place(x=30, y=35)  
 id\_combo = Combobox(update\_window)  
 id\_combo['values'] = get\_ID(flag)  
 id\_combo.place(x=200, y=35)  
  
 value\_label = Label(update\_window, text='Значение', font=('Arial Bold', 10))  
 value\_label.place(x=30, y=90)  
 value\_textbox = Entry(update\_window, width=15)  
 value\_textbox.place(x=200, y=90)  
  
 date\_label = Label(update\_window, text='Дата', font=('Arial Bold', 10))  
 date\_label.place(x=30, y=145)  
 date\_textbox = Entry(update\_window, width=15)  
 date\_textbox.place(x=200, y=145)  
  
 category\_label = Label(update\_window, text='Категория', font=('Arial Bold', 10))  
 category\_label.place(x=30, y=200)  
 category\_combo = Combobox(update\_window)  
 if not flag:  
 category\_combo['values'] = get\_categories\_i()  
 else:  
 category\_combo['values'] = get\_categories\_e()  
 category\_combo.place(x=200, y=200)  
  
 def update():  
 id = id\_combo.get()  
 value = value\_textbox.get()  
 date = date\_textbox.get()  
 category = category\_combo.get()  
 if id != '' and value != '' and date != '' and category != '':  
 if all(i.isdigit() for i in value):  
 date\_regex = re.compile(r'^\d{4}-\d{2}-\d{2}$')  
 if date\_regex.match(date):  
 update\_i\_e(id, value, date, category, flag)  
 messagebox.showinfo('Изменение', f'Операция изменена (ID={id})')  
 update\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Неверный формат даты. (Правильный: "yyyy-mm-dd")')  
 else:  
 messagebox.showerror('Ошибка', 'Введено неверное значение.')  
 else:  
 messagebox.showerror('Ошибка', 'Все поля должны быть заполнены (ID, значение, дата, категория).')  
  
 add\_button = Button(update\_window, text="Изменить", height=3, width=20, command=update)  
 add\_button.place(x=110, y=250)  
  
 change\_category\_button = Button(update\_window, text="\\", height=1, width=3, command=lambda: change\_category(flag))  
 change\_category\_button.place(x=355, y=200)  
  
 update\_window.mainloop()  
  
  
def update\_moneybox():  
 moneybox\_window = Tk()  
 moneybox\_window.title('Изменить запись')  
 moneybox\_window.geometry('400x325')  
 moneybox\_window.resizable(False, False)  
 moneybox\_window.focus\_set()  
 moneybox\_window.grab\_set()  
  
 id\_label = Label(moneybox\_window, text='ID', font=('Arial Bold', 10))  
 id\_label.place(x=30, y=35)  
 id\_combo = Combobox(moneybox\_window)  
 id\_combo['values'] = get\_ID\_r()  
 id\_combo.place(x=200, y=35)  
  
 value\_label = Label(moneybox\_window, text='Значение', font=('Arial Bold', 10))  
 value\_label.place(x=30, y=90)  
 value\_textbox = Entry(moneybox\_window, width=15)  
 value\_textbox.place(x=200, y=90)  
  
 date\_label = Label(moneybox\_window, text='Дата', font=('Arial Bold', 10))  
 date\_label.place(x=30, y=145)  
 date\_textbox = Entry(moneybox\_window, width=15)  
 date\_textbox.place(x=200, y=145)  
  
 def update():  
 id = id\_combo.get()  
 value = value\_textbox.get()  
 date = date\_textbox.get()  
 if value != '' and date != '' and id != '':  
 if all(i.isdigit() for i in value) or value[0] == '-' and all(i.isdigit() for i in value[1:]):  
 date\_regex = re.compile(r'^\d{4}-\d{2}-\d{2}$')  
 if date\_regex.match(date):  
 update\_r(id, value, date)  
 messagebox.showinfo('Изменение', f'Операция изменена (ID={id})')  
 moneybox\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'Неверный формат даты. (Правильный: "yyyy-mm-dd")')  
 else:  
 messagebox.showerror('Ошибка', 'Введено неверное значение.')  
 else:  
 messagebox.showerror('Ошибка', 'Все поля должны быть заполнены (ID, значение, дата).')  
  
 add\_button = Button(moneybox\_window, text="Изменить", height=3, width=20, command=update)  
 add\_button.place(x=110, y=250)  
  
 moneybox\_window.mainloop()  
  
  
def delete\_operation(flag):  
 delete\_window = Tk()  
 delete\_window.title('Удалить запись')  
 delete\_window.geometry('400x200')  
 delete\_window.resizable(False, False)  
 delete\_window.focus\_set()  
 delete\_window.grab\_set()  
  
 id\_label = Label(delete\_window, text='ID записи', font=('Arial Bold', 15))  
 id\_label.place(x=50, y=50)  
 id\_combo = Combobox(delete\_window)  
 if flag == 0:  
 id\_combo['values'] = get\_ID(0)  
 elif flag == 1:  
 id\_combo['values'] = get\_ID(1)  
 else:  
 id\_combo['values'] = get\_ID\_r()  
 id\_combo.place(x=200, y=55)  
  
 def delete\_o():  
 id = id\_combo.get()  
 if id != '':  
 delete\_i\_e\_r(id, flag)  
 messagebox.showinfo('Удаление', f'Операция удалена (ID={id}).')  
 delete\_window.destroy()  
 else:  
 messagebox.showerror('Ошибка', 'ID операции не указан.')  
  
 add\_button = Button(delete\_window, text="Удалить", height=2, width=15, command=delete\_o)  
 add\_button.place(x=115, y=130)  
  
 delete\_window.mainloop()  
  
  
def main\_window\_user(userID):  
 first\_window.destroy()  
 main\_window = Tk()  
 main\_window.title('Домашние финансы')  
 main\_window.geometry('800x450')  
 main\_window.resizable(False, False)  
 main\_window.focus\_set()  
  
 main\_control = ttk.Notebook(main\_window)  
 budget = ttk.Frame(main\_control)  
 incomes = ttk.Frame(main\_control)  
 expenses = ttk.Frame(main\_control)  
 momeybox = ttk.Frame(main\_control)  
 main\_control.add(budget, text='Баланс')  
 main\_control.add(incomes, text='Доходы')  
 main\_control.add(expenses, text='Расходы')  
 main\_control.add(momeybox, text='Копилка')  
  
 balance\_label\_text = Label(budget, text='Баланс', font=('Arial Bold', 25))  
 balance\_label\_text.pack(side=TOP, pady=40)  
 balance\_label = Entry(budget, font=('Arial Bold', 25), justify='center')  
 balance\_label.insert(END, get\_balance())  
 balance\_label.pack(side=TOP, pady=50)  
  
 def set\_text\_by\_button():  
 balance\_label.delete(0, "end")  
 balance\_label.insert(0, get\_balance())  
 balance\_button = Button(budget, text="Обновить", height=3, width=20, command=set\_text\_by\_button)  
 balance\_button.pack(anchor=S, pady=50)  
  
 income\_label = Label(incomes, text='Поступления', font=('Arial Bold', 30))  
 income\_label.pack(side=TOP, pady=30)  
 add\_income\_button = Button(incomes, text="Пополнить", height=3, width=20, command=lambda: add\_income(userID))  
 add\_income\_button.place(x=180, y=350)  
 income\_table = show\_table(incomes, True)  
 income\_table.pack()  
  
 def refresh\_incomes():  
 nonlocal income\_table  
 income\_table.destroy()  
 income\_table = show\_table(incomes, True)  
 income\_table.pack()  
  
 income\_table\_button = Button(incomes, text="Обновить", height=3, width=20, command=refresh\_incomes)  
 income\_table\_button.place(x=440, y=350)  
 if get\_user\_status(userID) == 1:  
 income\_update\_button = Button(incomes, text="\\", height=1, width=3,  
 command=lambda: update\_operation(False))  
 income\_update\_button.place(x=730, y=120)  
 income\_delete\_button = Button(incomes, text="x", height=1, width=3,  
 command=lambda: delete\_operation(0))  
 income\_delete\_button.place(x=730, y=150)  
  
 expense\_label = Label(expenses, text='Затраты', font=('Arial Bold', 30))  
 expense\_label.pack(side=TOP, pady=30)  
 add\_expense\_button = Button(expenses, text="Заплатить", height=3, width=20, command=lambda: add\_expense(userID))  
 add\_expense\_button.place(x=180, y=350)  
 expense\_table = show\_table(expenses, False)  
 expense\_table.pack()  
  
 def refresh\_expenses():  
 nonlocal expense\_table  
 expense\_table.destroy()  
 expense\_table = show\_table(expenses, False)  
 expense\_table.pack()  
  
 expense\_table\_button = Button(expenses, text="Обновить", height=3, width=20, command=refresh\_expenses)  
 expense\_table\_button.place(x=440, y=350)  
 if get\_user\_status(userID) == 1:  
 expense\_update\_button = Button(expenses, text="\\", height=1, width=3,  
 command=lambda: update\_operation(True))  
 expense\_update\_button.place(x=730, y=120)  
 expense\_delete\_button = Button(expenses, text="x", height=1, width=3,  
 command=lambda: delete\_operation(1))  
 expense\_delete\_button.place(x=730, y=150)  
  
 moneybox\_label = Label(momeybox, text='Баланс копилки', font=('Arial Bold', 25))  
 moneybox\_label.place(x=80, y=30)  
 moneybox\_textbox = Entry(momeybox, font=('Arial Bold', 20), justify='center')  
 moneybox\_textbox.insert(END, get\_balance\_r())  
 moneybox\_textbox.place(x=40, y=150)  
 moneybox\_table = show\_table\_moneybox(momeybox)  
 moneybox\_table.place(x=370, y=30)  
  
 def set\_text\_by\_button\_r():  
 moneybox\_textbox.delete(0, "end")  
 moneybox\_textbox.insert(0, get\_balance\_r())  
  
 nonlocal moneybox\_table  
 moneybox\_table.destroy()  
 moneybox\_table = show\_table\_moneybox(momeybox)  
 moneybox\_table.place(x=370, y=30)  
  
 balance\_button = Button(momeybox, text="Обновить", height=3, width=20, command=set\_text\_by\_button\_r)  
 balance\_button.place(x=100, y=250)  
  
 add\_moneybox\_button = Button(momeybox, text="Пополнить", height=3, width=20,  
 command=lambda: add\_moneybox(userID, True))  
 add\_moneybox\_button.place(x=400, y=350)  
 take\_moneybox\_button = Button(momeybox, text="Снять", height=3, width=20,  
 command=lambda: add\_moneybox(userID, False))  
 take\_moneybox\_button.place(x=600, y=350)  
 if get\_user\_status(userID) == 1:  
 moneybox\_update\_button = Button(momeybox, text="\\", height=1, width=3,  
 command=lambda: update\_moneybox())  
 moneybox\_update\_button.place(x=330, y=220)  
 moneybox\_delete\_button = Button(momeybox, text="x", height=1, width=3,  
 command=lambda: delete\_operation(2))  
 moneybox\_delete\_button.place(x=330, y=250)  
  
 main\_control.pack(expand=1, fill='both')  
 main\_window.mainloop()  
  
  
first\_window = Tk()  
first\_window.title("Вход/регистрация")  
first\_window.geometry('800x450')  
first\_window.resizable(False, False)  
first\_window.focus\_set()  
  
tab\_control = ttk.Notebook(first\_window)  
enter = ttk.Frame(tab\_control)  
registration = ttk.Frame(tab\_control)  
tab\_control.add(enter, text='Вход')  
tab\_control.add(registration, text='Регистрация')  
  
  
hello\_label = Label(enter, text="Добро пожаловать", font=("Arial Bold", 30))  
hello\_label.pack(side=TOP, pady=20)  
name\_label = Label(enter, text="Пользователь", font=("Arial Bold", 15))  
name\_label.place(x=120, y=170)  
password\_label = Label(enter, text="Пароль", font=("Arial Bold", 15))  
password\_label.place(x=120, y=225)  
name\_combo = Combobox(enter)  
name\_combo['values'] = get\_users()  
name\_combo.place(x=500, y=170)  
  
  
def update\_name\_combo():  
 global name\_combo  
 name\_combo['values'] = get\_users()  
  
  
update\_name = Button(enter, text="[]", height=1, width=3, command=update\_name\_combo)  
update\_name.place(x=700, y=170)  
password\_textbox = Entry(enter, width=23)  
password\_textbox.place(x=500, y=227)  
enter\_button = Button(enter, text="Вход", height=3, width=20, command=check\_enter)  
enter\_button.pack(side=BOTTOM)  
  
registration\_label = Label(registration, text="Регистрация", font=("Arial Bold", 30))  
registration\_label.pack(side=TOP, pady=20)  
new\_name\_label = Label(registration, text="ФИО", font=("Arial Bold", 15))  
new\_name\_label.place(x=140, y=160)  
new\_name\_textbox = Entry(registration, width=23)  
new\_name\_textbox.place(x=500, y=160)  
new\_password\_label = Label(registration, text="Пароль", font=("Arial Bold", 15))  
new\_password\_label.place(x=140, y=205)  
new\_password\_textbox = Entry(registration, width=23)  
new\_password\_textbox.place(x=500, y=205)  
status\_label = Label(registration, text="Статус", font=("Arial Bold", 15))  
status\_label.place(x=140, y=250)  
status\_combo = Combobox(registration)  
status\_combo['values'] = ('Пользователь', 'Администратор')  
status\_combo.place(x=500, y=250)  
new\_user\_button = Button(registration, text="Добавить", height=3, width=20, command=add\_user)  
new\_user\_button.pack(side=BOTTOM)  
  
tab\_control.pack(expand=1, fill='both')  
first\_window.mainloop()  
  
  
if connection:  
 cursor.close()  
 connection.close()  
 print("Соединение с PostgreSQL закрыто")

Описание кода:

* Импортирование всех необходимых библиотек и методов;
* Функция «add\_user»

Добавляет нового пользователя, совершает проверки введённых данных;

* Функция «check\_enter»

Осуществляет проверку пользователя при входе в приложение, открывает основное окно приложения;

* Функция «show\_table»

Создаёт визуальное представление таблиц доходов, расходов;

* Функция «show\_table\_moneybox»

Создаёт визуальное представление таблицы переводов в копилке;

* Функция «add\_income»

Создаёт окно добавления новой записи о доходе, делает все необходимые проверки;

* Функция «add\_expense»

Создаёт окно добавления новой записи о расходе, делает все необходимые проверки;

* Функция «add\_moneybox»

Создаёт окно добавления новой записи о переводе в копилку или из неё, делает все необходимые проверки;

* Функция «change\_category»

Создаёт окно для редактирования записей категорий, даёт возможность создать новую категорию или удалить существующую, осуществляет все необходимые проверки;

* Функция «update\_operation»

Создаёт окно редактирования записей о доходах или расходах, даёт возможность изменить запись по её ID, осуществляет все необходимые проверки;

* Функция «update\_moneybox»

Создаёт окно редактирования записей о переводах в копилке, даёт возможность изменить запись по её ID, осуществляет все необходимые проверки;

* Функция «delete\_operation»

Создаёт окно для удаления записей о доходах, расходах или переводах, осуществляет все необходимые проверки;

* Функция «main\_window\_user»

Создаёт основное окно, где отображается баланс, записи о доходах, расходах или переводах, вызывает все функции для создания, редактирования и удаления записей;

* Создание первого окна «first\_window», на котором осуществляется регистрация и вход в приложение, находится вне функции;
* При закрытии оконного приложения соединение с базой данных закрывается.

1. **Демонстрация интерфейса.**

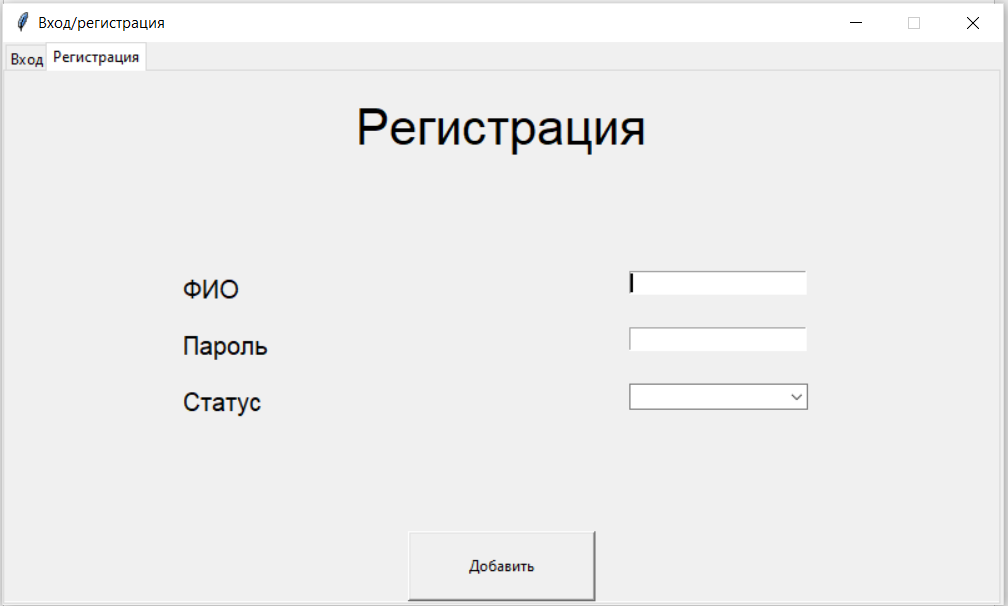


Рисунок 27 – Окно регистрации пользователя

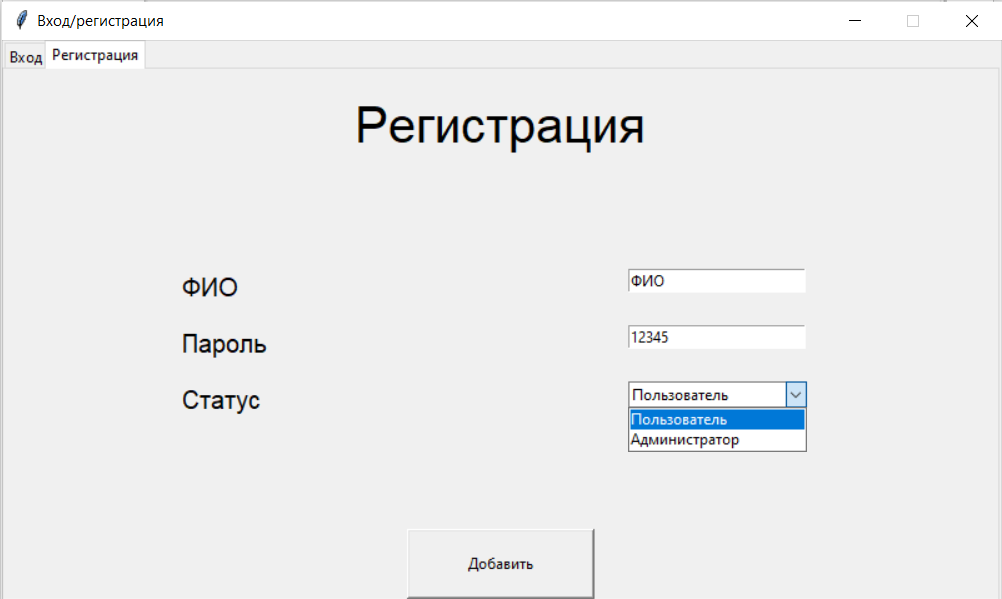


Рисунок 28 – Заполненная форма регистрации пользователя

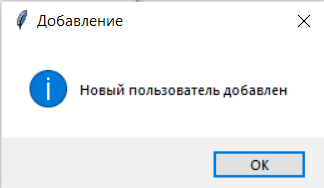


Рисунок 29 – Уведомление о добавлении нового пользователя

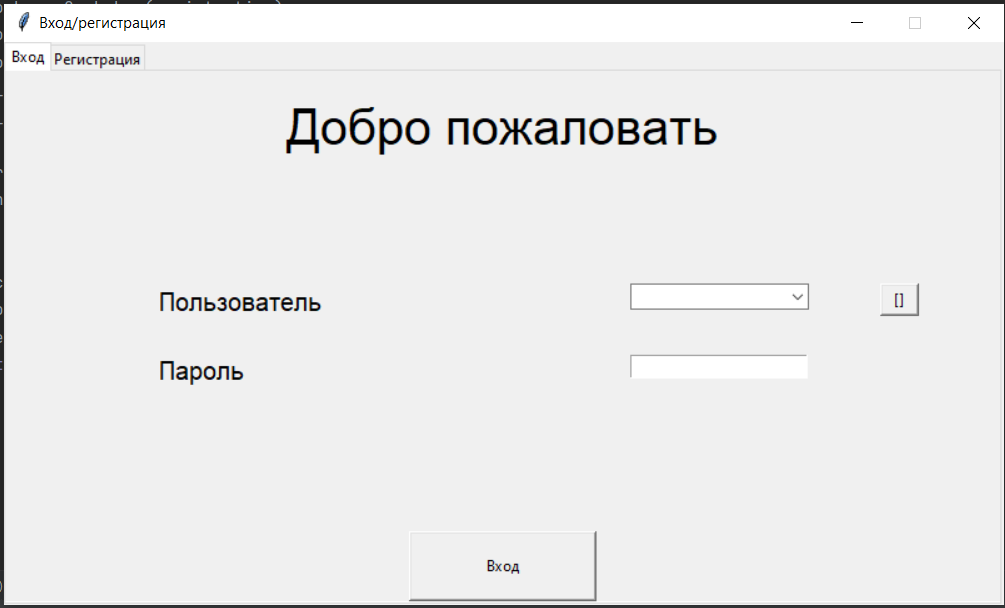


Рисунок 30 – Окно входа в приложение

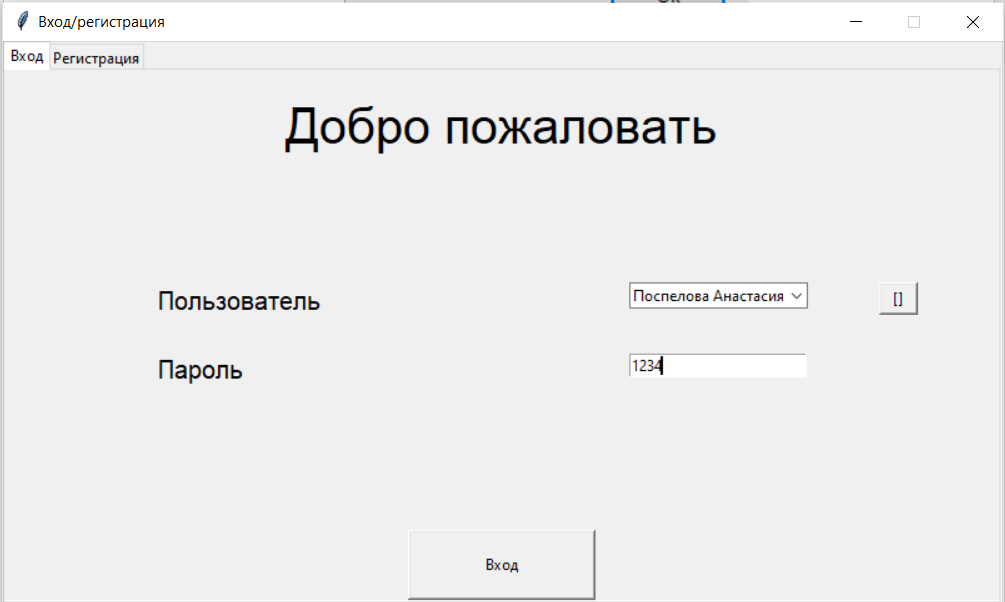


Рисунок 31 – Заполненная форма входа в приложение

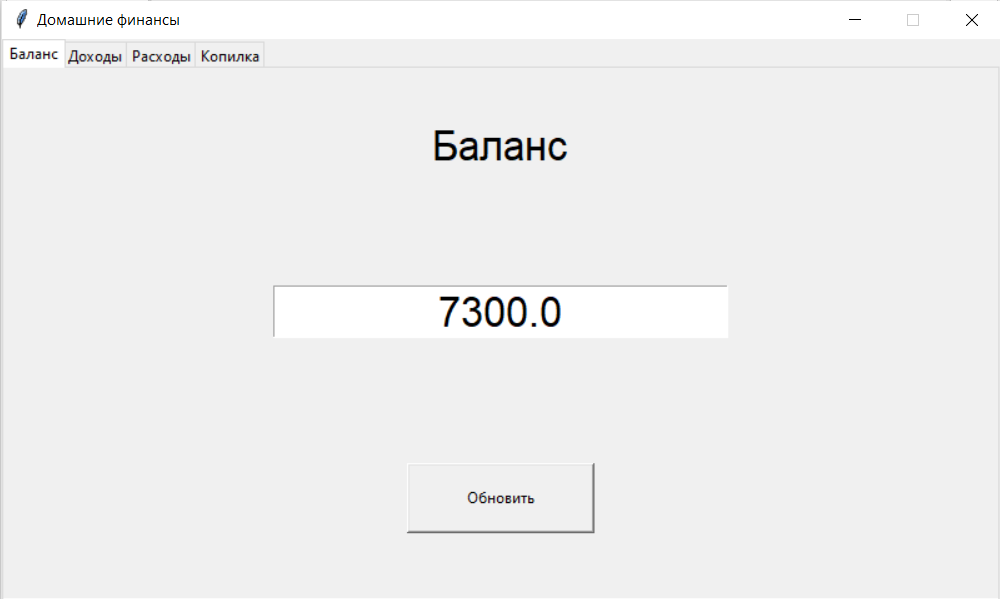


Рисунок 32 – Основное окно, баланс бюджета

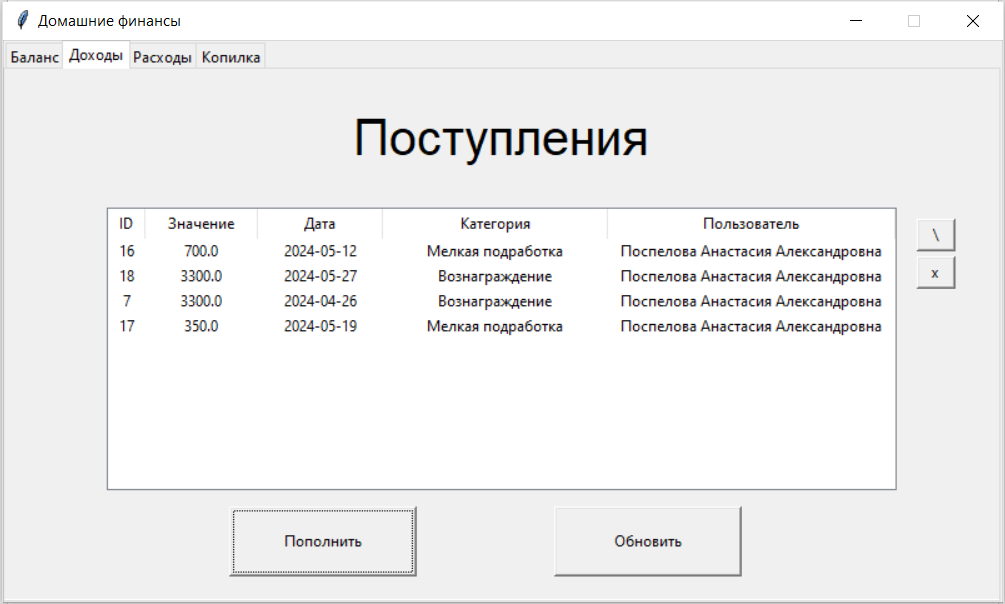


Рисунок 33 – Окно записей о доходах

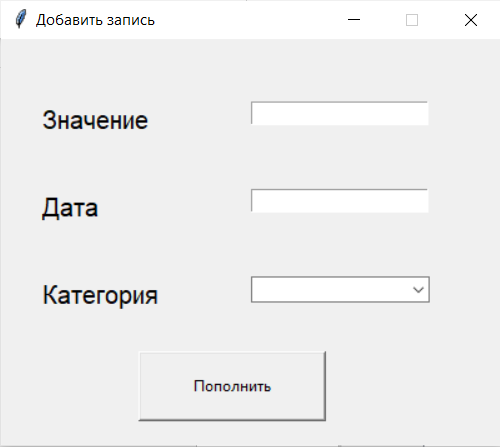


Рисунок 34 – Окно новой записи о доходе

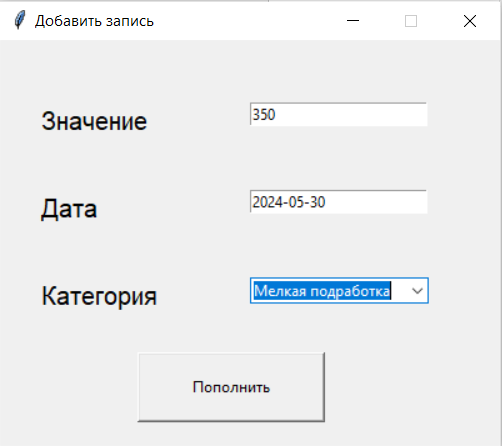


Рисунок 35 – Заполненная форма новой записи о доходе

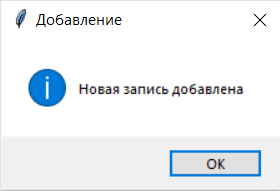


Рисунок 36 – Окно подтверждения о добавлении новой записи о доходе

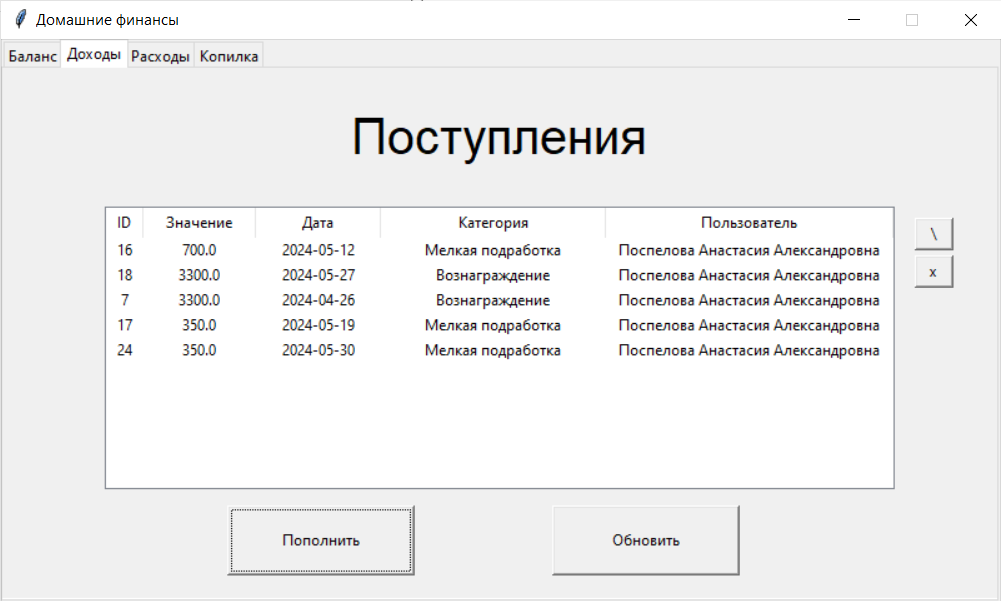


Рисунок 37 – Обновлённое окно записей о доходах с только что добавленной записью

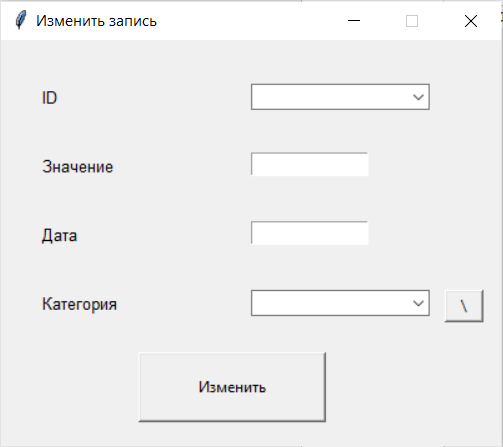


Рисунок 38 – Окно редактирования записи о доходе

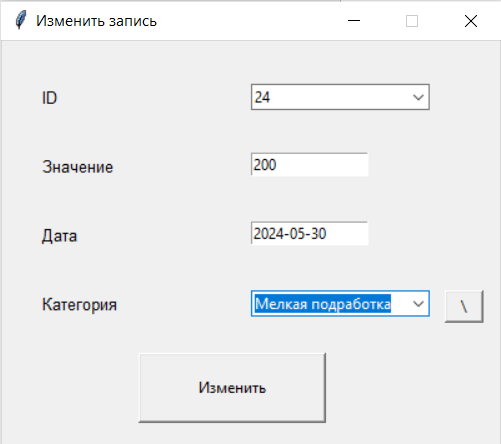


Рисунок 39 – Заполненная форма редактирования записи о доходе

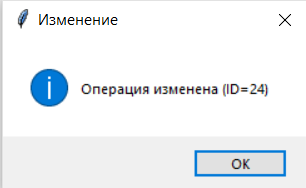


Рисунок 40 – Окно уведомления об изменении записи

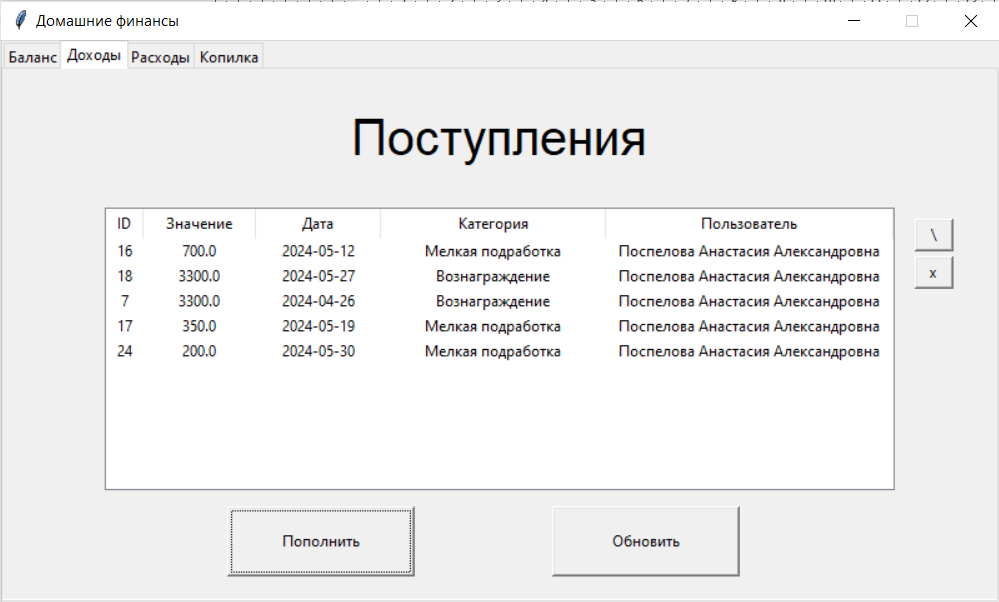


Рисунок 41 - Обновлённое окно записей о доходах с только что изменённой записью

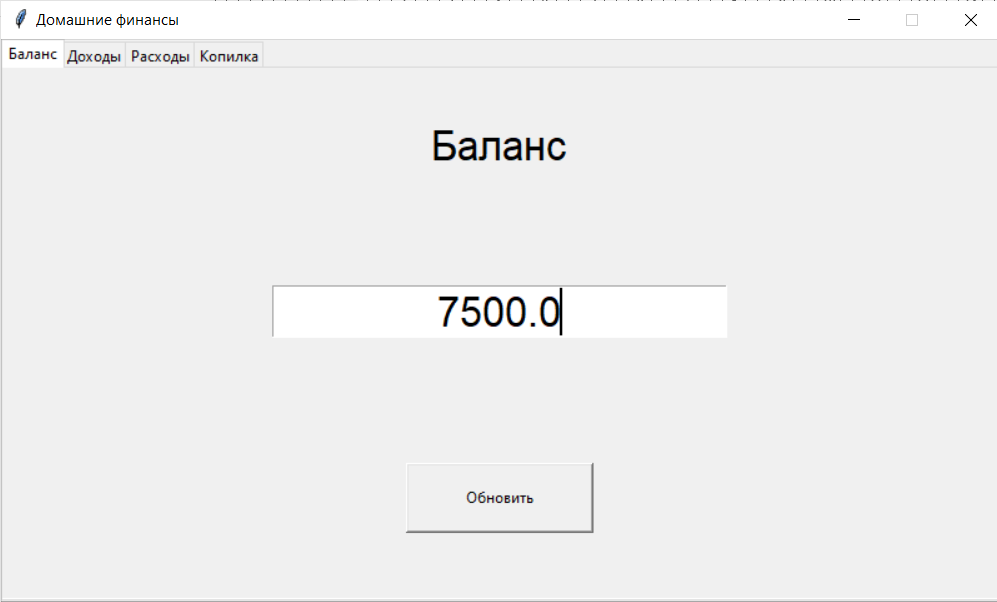


Рисунок 42 – Обновлённое окно баланса после всех изменений

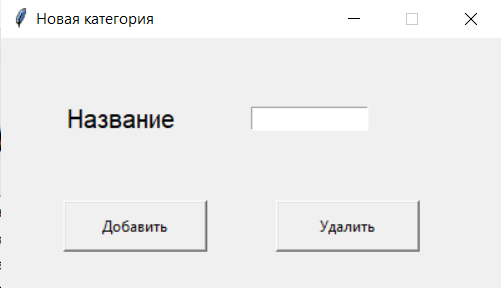


Рисунок 43 – Окно редактирования категорий

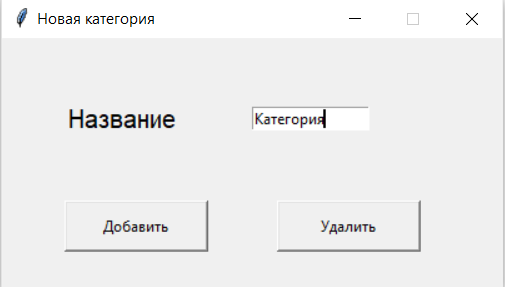


Рисунок 44 – Заполненная форма редактирования категорий

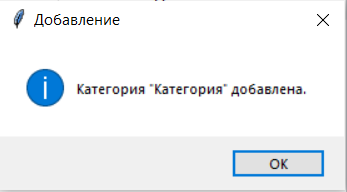


Рисунок 45 – Окно уведомления о добавлении новой категории

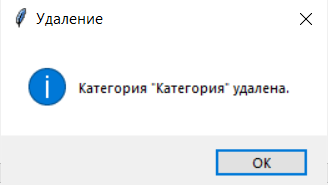


Рисунок 46 – Окно уведомления об удалении категории

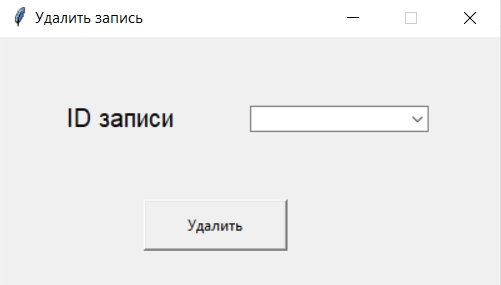


Рисунок 47 – Окно удаления записи о доходе

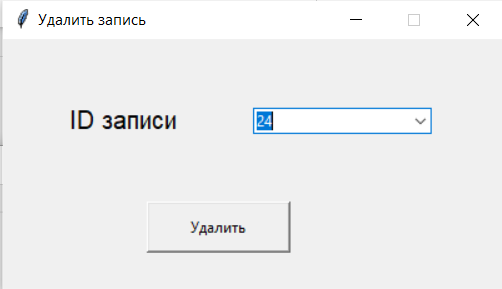


Рисунок 48 – Заполненная форма удаления записи о доходе

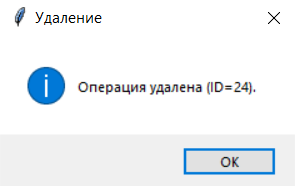


Рисунок 49 – Окно уведомления об удалении записи о доходе

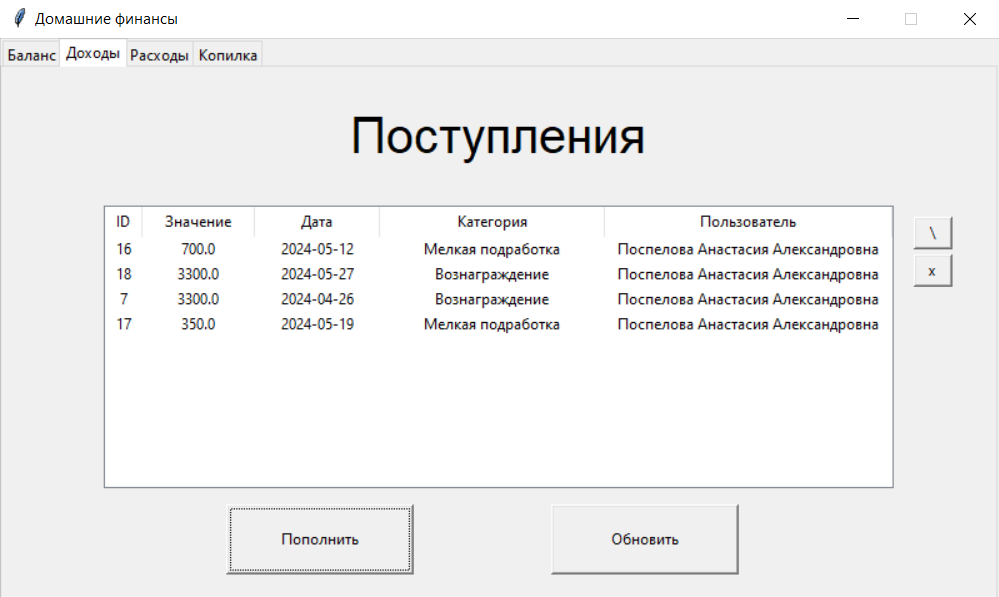


Рисунок 50 – Обновлённое окно записей о доходах без только что удалённой записи

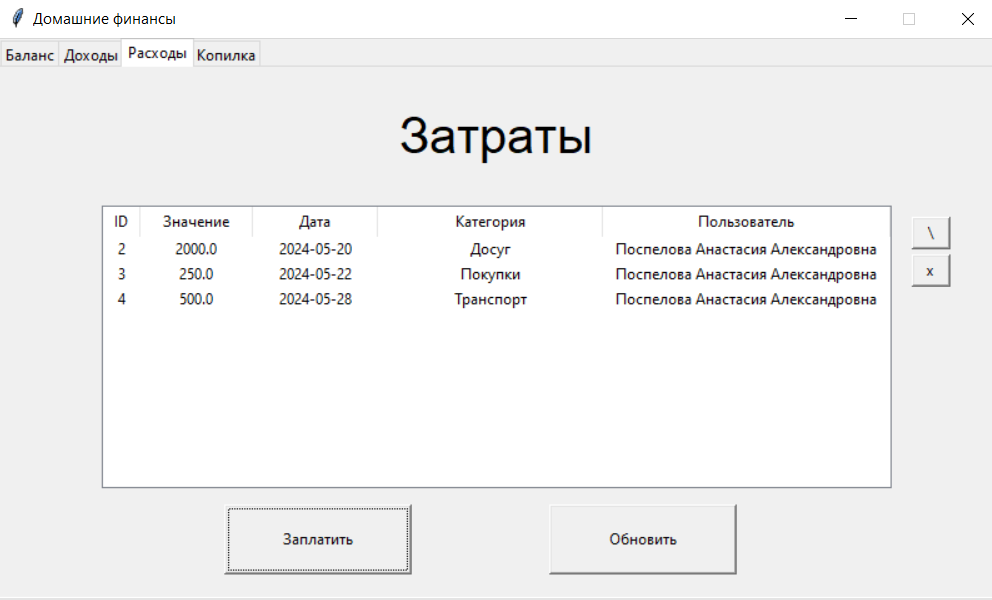


Рисунок 51 – Окно записей о расходах (работает по аналогии с окном записей о доходах)

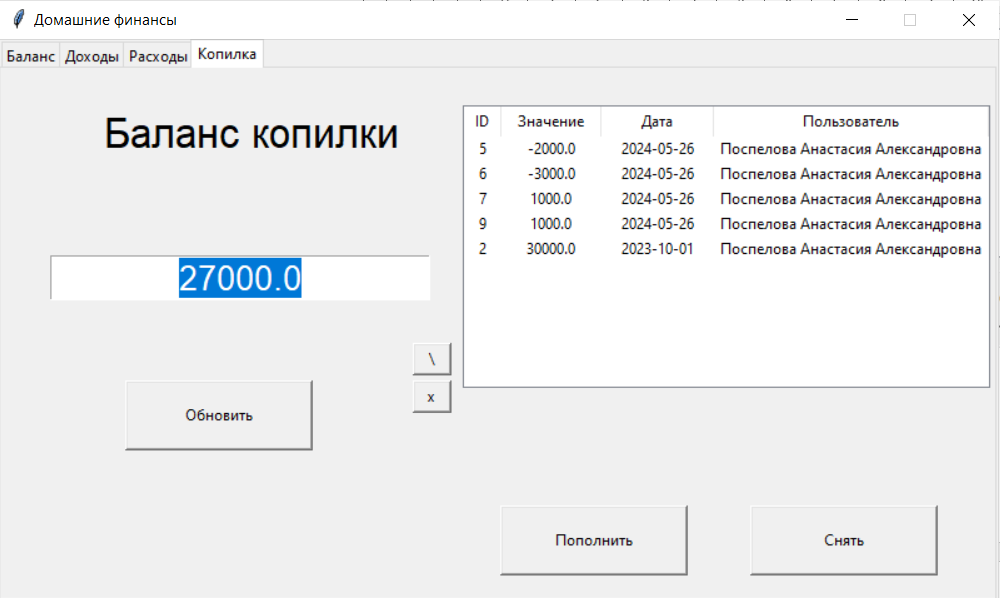


Рисунок 52 – Окно записей о переводах копилки (работает по аналогии с окном записей о доходах)

**ЗАКЛЮЧЕНИЕ**

Результатом курсового проекта является реализованная программа для работы с информационной системой «Агентство недвижимости». Приложение позволяет добавлять, удалять, просматривать и искать данные о клиентах, менеджерах, объектах недвижимости и сделках. В программе обработаны основные исключительные ситуации. Программа состоит из связанных между собой взаимодействующих классов.

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Документация по psycopg:

[https://www.psycopg.org/](https://www.psycopg.org/docs/module.html)

1. Документация по tkinter:

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