

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Ярославский государственный технический университет»  
Кафедра «Информационные системы и технологии»

Работа защищена  
с оценкой \_\_\_\_\_  
Преподаватель  
\_\_\_\_\_ А.Н. Прозоров  
«\_\_»\_\_\_\_\_ 2025

## **КНИГА УЧЕТА С SQLLITE**

Отчет по лабораторной работе №2  
по курсу «Технологии программирования»

ЯГТУ 09.03.02-001 ЛР

Работу выполнила  
студентка группы ЗЦИС-26  
\_\_\_\_\_ А.Е.Петрова  
«\_\_»\_\_\_\_\_ 2025

### **Цель работы:**

1. Сформировать и внедрить базу данных SQLite, состоящую из сущностей, соответствующих проекту (категории, операции).
2. Реализовать загрузку данных книги (категорий и транзакций) из базы данных SQLite при запуске проекта.
3. Реализовать возможность загрузки книги из файла JSON с возможностью перезаписи текущих данных или их дополнения.

### **Задание:**

Сформировать и внедрить БД SQLite, состоящей из сущностей, удовлетворяющих проекту (категории, операции). При запуске проекта, база данных должна передавать данные книги. В случае загрузки книги из файла должен открываться диалог, предлагающий перезаписать текущую книгу, либо её дополнить

Выполненные задачи и результаты:

Создание базы данных SQLite:

Создана база данных SQLite с именем «accounting.db».

Разработана схема базы данных, состоящая из двух таблиц: «Категории» и «Транзакции».

Таблица «Категории» содержит поля:

- «Название» (ТЕКСТ, первичный ключ, название категории);
- «Описание» (ТЕКСТ, описание категории).

Таблица «Транзакции» содержит поля:

- «Дата» (TEXT, дата операции),
- «Описание» (TEXT, описание операции),
- «Сумма» (REAL, сумма операции),
- «Название категории» (TEXT, внешний ключ, ссылающийся на «Категории» (Name), название категории),
- «Тип» (INTEGER, тип операции: 0 — расход, 1 — доход).

Реализован класс DatabaseHelper для работы с базой данных: создание таблиц, добавление, обновление, удаление и чтение данных из таблиц «Категории» и «Транзакции».

Загрузка данных из базы данных при запуске проекта:

В методе Form1\_Load главного окна приложения выполняется проверка наличия файла базы данных “accounting.db”.

Если файл базы данных не существует, выполняется попытка загрузить начальные данные из файла JSON («initial\_data.json»).

Создается экземпляр класса DatabaseHelper для подключения к базе данных.

Создается экземпляр класса AccountingData, который содержит списки категорий и транзакций в памяти.

Загрузка данных из базы выполняется в методе Form1\_Load с помощью методов класса DatabaseHelper ( GetAllCategories, GetAllTransactions ) и сохраняется в списках Categories и Transactions класса AccountingData.

Списки Categories и Transactions класса AccountingData привязываются к элементам управления DataGridView для отображения данных в главном окне приложения.

Реализация загрузки данных из файла JSON с возможностью перезаписи или дополнения:

При нажатии на кнопку «Загрузить» открывается стандартное диалоговое окно выбора файла (OpenFileDialog) с фильтром для файлов JSON.

После выбора файла пользователю предлагается диалоговое окно с выбором действия:

«Да» — перезаписать текущие данные (очистить базу данных и загрузить данные из файла).

«Нет» — дополнить текущие данные (добавить новые категории и транзакции из файла, не удаляя существующие).

Реализована логика перезаписи и дополнения данных:

Перезапись:

- Метод `ClearAllData` класса `DatabaseHelper` очищает таблицы «Категории» и «Транзакции» в базе данных.
- Списки `Categories` и `Transactions` класса `AccountingData` очищаются.
- Данные из JSON-файла десериализуются в объект `AccountingData`.
- Категории и транзакции из десериализованного объекта добавляются в базу данных и в списки класса `AccountingData`.

Дополнение:

Данные из JSON-файла десериализуются в объект `AccountingData`.

Перед добавлением каждой категории и транзакции выполняется проверка на наличие дубликатов:

Проверяется, существует ли категория (или транзакция) с такими же параметрами в базе данных.

Категория (или транзакция) добавляется только в том случае, если она не существует в базе данных.

Новые категории и транзакции добавляются в базу данных и в списки класса `AccountingData`.

Содержание файлов приложения представлено в приложениях А, Б, В, Г.

**Вывод:**

В ходе выполнения лабораторной работы были получены навыки работы с базой данных SQLite и библиотекой Newtonsoft.Json. Разработана система учета личных финансов, позволяющая загружать данные как из базы данных SQLite, так и из файла JSON. Реализована возможность перезаписи и дополнения данных при загрузке из файла JSON.

## Содержимое файла Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using Книга_учета;
using Newtonsoft.Json;
using System.IO;
using System.Windows.Forms.DataVisualization.Charting;
using System.Data.SQLite;

namespace Книга_учета
{
    public partial class Form1 : Form
    {
        private AccountingData accountingData;
        private BindingSource categoriesBindingSource = new BindingSource();
        private BindingSource transactionsBindingSource = new BindingSource();
        private DatabaseHelper dbHelper;
        private string dbFilePath = "accounting.db";
        private string jsonFilePath = "initial_data.json"; // Путь к файлу JSON по
умолчанию

        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            // 1. Инициализация DatabaseHelper и создание/открытие БД
            dbHelper = new DatabaseHelper(dbFilePath);
            dbHelper.CreateDatabaseIfNotExists();

            Console.WriteLine($"dbHelper initialized: {dbHelper != null}"); //
Добавили вывод

            // 2. Инициализация AccountingData
            accountingData = new AccountingData();
        }
    }
}
```

```

accountingData.Categories = dbHelper.GetAllCategories();
accountingData.Transactions = dbHelper.GetAllTransactions();

Console.WriteLine($"accountingData initialized: {accountingData !=
null}"); // Добавили вывод

// 3. Привязка данных к DataGridView
categoriesBindingSource.DataSource = accountingData.Categories;
dgvCategories.DataSource = categoriesBindingSource;
dgvCategories.AutoGenerateColumns = false;
SetupCategoriesDataGridViewColumns(); // Настройка колонок

transactionsBindingSource.DataSource = accountingData.Transactions;
dgvTransactions.DataSource = transactionsBindingSource;
dgvTransactions.AutoGenerateColumns = false;
SetupTransactionsDataGridViewColumns(); // Настройка колонок

// 4. Настройка DataGridView (обработка ошибок, редактирование и
т.д.)
dgvTransactions.DataError += dgvTransactions_DataError;
dgvTransactions.SelectionMode                               =
DataGridViewSelectionMode.FullRowSelect; // Выделяем строку целиком
dgvTransactions.MultiSelect = false; // Запрещаем множественный
выбор
dgvTransactions.SelectionChanged                             +=
dgvTransactions_SelectionChanged; // Подписываемся на SelectionChanged

// 5. Настройка фильтров для OpenFileDialog и SaveFileDialog
openFileDialog1.Filter = "JSON files (*.json)|*.json|All files (*.*)|*.*";
saveFileDialog1.Filter = "JSON files (*.json)|*.json|All files (*.*)|*.*";

// 6. Загрузка данных при запуске (если нужно)
LoadDataFromFileOnStartup();

// 8. Обновление интерфейса (баланс, итого по категориям, график)
UpdateBalance();
UpdateCategoryTotals();
UpdateChart();
UpdateCategoryComboBox(); // Обновляем ComboBox категорий
}

private void SetupCategoriesDataGridViewColumns()
{
    dgvCategories.Columns.Clear();

```

```

        // Column Name
        DataGridViewTextBoxColumn nameColumn = new
DataGridViewTextBoxColumn();
        nameColumn.DataPropertyName = "Name";
        nameColumn.HeaderText = "Название";
        nameColumn.Name = "Name";
        dgvCategories.Columns.Add(nameColumn);

        // Column Description
        DataGridViewTextBoxColumn descriptionColumn = new
DataGridViewTextBoxColumn();
        descriptionColumn.DataPropertyName = "Description";
        descriptionColumn.HeaderText = "Описание";
        descriptionColumn.Name = "Description";
        dgvCategories.Columns.Add(descriptionColumn);

        // Запрещаем редактирование ячеек напрямую
        foreach (DataGridViewColumn column in dgvCategories.Columns)
        {
            column.ReadOnly = true;
        }
        dgvCategories.AllowUserToAddRows = false; //Отключаем добавление
строк
        dgvCategories.AllowUserToDeleteRows = false; //Отключаем удаление
строк
    }

    private void SetupTransactionsDataGridViewColumns()
    {
        dgvTransactions.Columns.Clear();

        // Column Date
        DataGridViewTextBoxColumn dateColumn = new
DataGridViewTextBoxColumn();
        dateColumn.DataPropertyName = "Date";
        dateColumn.HeaderText = "Дата";
        dateColumn.Name = "Date";
        dateColumn.DefaultCellStyle.Format = "yyyy-MM-dd"; // Format date
        dgvTransactions.Columns.Add(dateColumn);

        // Column Description
        DataGridViewTextBoxColumn descriptionColumn = new
DataGridViewTextBoxColumn();
        descriptionColumn.DataPropertyName = "Description";
        descriptionColumn.HeaderText = "Описание";

```



```

descriptionColumn.Name = "Description";
dgvTransactions.Columns.Add(descriptionColumn);

// Column Amount
DataGridViewTextBoxColumn amountColumn = new
DataGridViewTextBoxColumn();
amountColumn.DataPropertyName = "Amount";
amountColumn.HeaderText = "Сумма";
amountColumn.Name = "Amount";
amountColumn.DefaultCellStyle.Format = "C"; // Format currency
dgvTransactions.Columns.Add(amountColumn);

// Column Category
DataGridViewTextBoxColumn categoryColumn = new
DataGridViewTextBoxColumn();
categoryColumn.DataPropertyName = "Category";
categoryColumn.HeaderText = "Категория";
categoryColumn.Name = "Category";
dgvTransactions.Columns.Add(categoryColumn);

// Column Type
DataGridViewTextBoxColumn typeColumn = new
DataGridViewTextBoxColumn();
typeColumn.DataPropertyName = "Type";
typeColumn.HeaderText = "Тип";
typeColumn.Name = "Type";
dgvTransactions.Columns.Add(typeColumn);

// Запрещаем редактирование ячеек напрямую
foreach (DataGridViewColumn column in dgvTransactions.Columns)
{
    column.ReadOnly = true;
}
dgvTransactions.AllowUserToAddRows = false; //Отключаем
добавление строк
dgvTransactions.AllowUserToDeleteRows = false; //Отключаем удаление
строк

}
private void LoadDataFromFileOnStartup()
{
    // Проверяем, существует ли база данных. Если нет, пытаемся
    загрузить данные из JSON.
    if (!File.Exists(dbFilePath))
    {

```

```

        if (File.Exists(jsonFilePath))
        {
            LoadDataFromFile(jsonFilePath, true);    // Загружаем и
перезаписываем
        }
    }

    private void dgvTransactions_DataError(object sender,
DataGridViewDataErrorEventArgs e)
    {
        string columnName = "";
        if (dgvTransactions.Columns.Count > 0 && e.ColumnIndex >= 0 &&
e.ColumnIndex < dgvTransactions.Columns.Count)
        {
            columnName = dgvTransactions.Columns[e.ColumnIndex].Name;    //
Получаем имя колонки
        }
        if (e.Exception is FormatException)
        {
            MessageBox.Show($"Некорректный формат данных в колонке
'{columnName}'. Пожалуйста, проверьте введенные данные.", "Ошибка
формата", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            e.ThrowException = false;
            e.Cancel = true;
        }
        else
        {
            MessageBox.Show($"Произошла ошибка в колонке '{columnName}':
{e.Exception.Message}", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            e.ThrowException = false;
        }
    }

    private void LoadDataFromFile()
    {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            string filePath = openFileDialog1.FileName;
            LoadDataFromFile(filePath, false);
        }
    }

    private void LoadDataFromFile(string filePath, bool isStartupLoad = false)

```

```

{
    try
    {
        // 1. Чтение файла
        string jsonData = null;
        try
        {
            jsonData = File.ReadAllText(filePath);
            Console.WriteLine($"JSON data read successfully:\n{jsonData}"); //
Выводим содержимое JSON в консоль
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при чтении файла:
{ex.Message}\n{ex.StackTrace}", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        // 2. Десериализация JSON
        AccountingData loadedData = null;
        try
        {
            loadedData =
JsonConvert.DeserializeObject<AccountingData>(jsonData);
            if (loadedData == null)
            {
                Console.WriteLine("DeserializeObject вернул NULL!");
//добавили вывод
            }
            else
            {
                Console.WriteLine($"Data after deserialization: Categories count =
{loadedData.Categories?.Count}, Transactions count =
{loadedData.Transactions?.Count}"); // Добавили вывод
            }
        }
        catch (JsonReaderException jex)
        {
            MessageBox.Show($"Ошибка при десериализации JSON
(JsonReaderException):
{jex.Message}\n{jex.Path}\n{jex.LineNumber}\n{jex.LinePosition}", "Ошибка
десериализации", MessageBoxButtons.OK, MessageBoxIcon.Error);

            return;
        }
    }
}

```

```

    }
    catch (Exception ex) // Обрабатываем другие возможные исключения
при десериализации
    {
        MessageBox.Show($"Ошибка при десериализации JSON
(Exception): {ex.Message}\n{ex.StackTrace}", "Ошибка десериализации",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    if (loadedData != null && loadedData.Categories != null &&
loadedData.Transactions != null)
    {
        DialogResult result = DialogResult.Yes; // По умолчанию
перезапись для initial load

        if (!isStartupLoad)
        {
            // Предлагаем пользователю выбор: перезаписать текущие
данные или добавить к текущим
            result = MessageBox.Show("Выберите действие:\nДа -
Перезаписать текущие данные\nНет - Добавить к текущим данным",
"Загрузка данных", MessageBoxButtons.YesNoCancel,
MessageBoxIcon.Question);
        }

        if (result == DialogResult.Yes)
        {
            // Перезаписать: очищаем текущие данные и загружаем новые
try
            {
                // Очищаем таблицы в базе данных
dbHelper.ClearAllData();

                // Очищаем списки в памяти
accountingData.Categories.Clear();
accountingData.Transactions.Clear();

                // Загружаем категории и транзакции из файла
foreach (var category in loadedData.Categories)
                {
                    dbHelper.AddCategory(category);
                    accountingData.AddCategory(category);
                }
            }
        }
    }

```

```

        foreach (var transaction in loadedData.Transactions)
        {
            dbHelper.AddTransaction(transaction);
            accountingData.AddTransaction(transaction);
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при перезаписи данных:
{ex.Message}\n{ex.StackTrace}", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }
}
else if (result == DialogResult.No)
{
    // Добавить: добавляем данные из файла к текущим
    try
    {
        // Добавляем категории, только если их еще нет в списке и в
базе данных
        foreach (var category in loadedData.Categories)
        {
            if (!accountingData.Categories.Any(c => c.Name ==
category.Name))
            {
                // Проверяем, есть ли такая категория в БД
                if (!dbHelper.GetAllCategories().Any(c => c.Name ==
category.Name))
                {
                    dbHelper.AddCategory(category);
                    accountingData.AddCategory(category);
                }
            }
        }

        // Добавляем транзакции, только если их еще нет в списке и в
базе данных
        foreach (var transaction in loadedData.Transactions)
        {
            // Проверяем, есть ли такая транзакция в БД
            if (!dbHelper.GetAllTransactions().Any(t =>
                t.Date == transaction.Date &&
                t.Description == transaction.Description &&

```

```

        t.Amount == transaction.Amount &&
        t.Category.Name == transaction.Category.Name &&
        t.Type == transaction.Type))
    {
        dbHelper.AddTransaction(transaction);
        accountingData.AddTransaction(transaction);
    }

    }
}
catch (Exception ex)
{
    MessageBox.Show($"Ошибка при добавлении данных:
{ex.Message}\n{ex.StackTrace}", "Ошибка",
    MessageBoxButtons.OK,
    MessageBoxIcon.Error);
    return;
}
}

// Обновляем BindingSource и UI
categoriesBindingSource.ResetBindings(false);
transactionsBindingSource.ResetBindings(false);
UpdateCategoryComboBox();
UpdateBalance();
UpdateCategoryTotals();
UpdateChart();
}
else
{
    MessageBox.Show("Ошибка: Не удалось загрузить данные из
файла. Возможно, файл поврежден или имеет неверный формат.", "Ошибка",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
catch (Exception ex)
{
    string message = $"Ошибка загрузки данных: {ex.Message}";
    if (ex.InnerException != null)
    {
        message += $"
Inner Exception: {ex.InnerException.Message}";
    }
    MessageBox.Show(message, "Ошибка",
    MessageBoxButtons.OK,
    MessageBoxIcon.Error);
}
}

```

```

    }

    private void SaveDataToFile()
    {
        if (saveFileDialog1.ShowDialog() == DialogResult.OK)
        {
            string filePath = saveFileDialog1.FileName;

            try
            {
                // Создаем объект для сериализации, содержащий категории и
транзакции
                var dataToSerialize = new
                {
                    Categories = accountingData.Categories,
                    Transactions = accountingData.Transactions
                };

                // Сериализуем данные в JSON
                string jsonData = JsonConvert.SerializeObject(dataToSerialize,
Newtonsoft.Json.Formatting.Indented);

                // Записываем JSON в файл
                File.WriteAllText(filePath, jsonData);

                MessageBox.Show("Данные успешно сохранены в файл.",
"Информация", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка при сохранении данных в файл:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }

    // Методы для категорий
    private void btnAddCategory_Click(object sender, EventArgs e)
    {
        string name = txtNameCategory.Text.Trim();
        string description = txtDescriptionCategory.Text.Trim();

        if (!string.IsNullOrEmpty(name))
        {
            // Добавляем категорию в базу данных и в список в памяти

```

```

        Category newCategory = new Category(name, description);
        dbHelper.AddCategory(newCategory);
        accountingData.AddCategory(newCategory);
        categoriesBindingSource.ResetBindings(false);
        UpdateCategoryComboBox();
        txtNameCategory.Clear();
        txtDescriptionCategory.Clear();
        UpdateCategoryTotals();
        UpdateChart();
    }
    else
    {
        MessageBox.Show("Введите название категории.",
            "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}

private void btnEditCategory_Click(object sender, EventArgs e)
{
    if (dgvCategories.SelectedRows.Count > 0)
    {
        Category selectedCategory =
            (Category)dgvCategories.SelectedRows[0].DataBoundItem;
        if (selectedCategory != null)
        {
            string oldName = selectedCategory.Name;
            string newName = txtNameCategory.Text.Trim();
            string newDescription = txtDescriptionCategory.Text.Trim();

            if (!string.IsNullOrEmpty(newName))
            {
                Category newCategory = new Category(newName,
                    newDescription);

                // Обновляем категорию в базе данных
                dbHelper.UpdateCategory(newCategory);

                // Обновляем категорию в списке в памяти
                accountingData.UpdateCategory(oldName, newCategory);

                categoriesBindingSource.ResetBindings(false);
                transactionsBindingSource.ResetBindings(false);
                UpdateCategoryComboBox();
                UpdateCategoryTotals();
                UpdateChart();
            }
        }
    }
}

```



```

        txtNameCategory.Clear();
        txtDescriptionCategory.Clear();
    }
    else
    {
        MessageBox.Show("Введите новое название категории.",
            "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}
}
else
{
    MessageBox.Show("Выберите категорию для редактирования.",
        "Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
}

private void btnDeleteCategory_Click(object sender, EventArgs e)
{
    if (dgvCategories.SelectedRows.Count > 0)
    {
        Category selectedCategory =
        (Category)dgvCategories.SelectedRows[0].DataBoundItem;
        if (selectedCategory != null)
        {
            DialogResult result = MessageBox.Show($"Вы уверены, что хотите
            удалить категорию '{selectedCategory.Name}'?", "Подтверждение удаления",
            MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (result == DialogResult.Yes)
            {
                // Удаляем категорию из базы данных
                dbHelper.DeleteCategory(selectedCategory.Name);

                // Удаляем категорию из списка в памяти
                accountingData.DeleteCategory(selectedCategory.Name);

                categoriesBindingSource.ResetBindings(false);
                transactionsBindingSource.ResetBindings(false);
                UpdateCategoryComboBox();
                UpdateCategoryTotals();
                UpdateChart();
            }
        }
    }
}

```

```

        else
        {
            MessageBox.Show("Выберите категорию для удаления.",
"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }

    private void UpdateCategoryComboBox()
    {
        cmbTransactionCategory.DataSource = null;
        cmbTransactionCategory.DataSource = accountingData.Categories;
        cmbTransactionCategory.DisplayMember = "Name";
        cmbTransactionCategory.ValueMember = "Name";
    }

    // Методы для транзакций
    private void btnAddTransaction_Click(object sender, EventArgs e)
    {
        if (cmbTransactionCategory.SelectedItem == null)
        {
            MessageBox.Show("Пожалуйста, выберите категорию.",
"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        Category selectedCategory =
(Category)cmbTransactionCategory.SelectedItem;

        // Проверяем, существует ли выбранная категория в списке категорий
        if (!accountingData.Categories.Contains(selectedCategory))
        {
            MessageBox.Show("Выбранная категория не существует.
Пожалуйста, выберите другую категорию.", "Предупреждение",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        DateTime date = dtpTransactionDate.Value.Date; // Сохраняем только
дату
        string description = txtTransactionDescription.Text.Trim();
        decimal amount = nudTransactionAmount.Value;

        TransactionType type = rdbTransactionExpense.Checked ?
TransactionType.Expense : TransactionType.Income; // Получаем тип из
RadioButton

```

```

        if (!string.IsNullOrEmpty(description))
        {
            Transaction transaction = new Transaction(date, description, amount,
selectedCategory, type);

            // Добавляем транзакцию в базу данных
            dbHelper.AddTransaction(transaction);

            // Добавляем транзакцию в список в памяти
            accountingData.AddTransaction(transaction);

            transactionsBindingSource.ResetBindings(false);
            UpdateBalance();
            UpdateCategoryTotals();
            UpdateChart();
            ClearTransactionFields();
        }
        else
        {
            MessageBox.Show("Пожалуйста,          введите          описание.",
"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }
    }

    private void btnEditTransaction_Click(object sender, EventArgs e)
    {
        if (dgvTransactions.SelectedRows.Count > 0)
        {
            Transaction          selectedTransaction          =
(Transaction)dgvTransactions.SelectedRows[0].DataBoundItem;
            if (selectedTransaction != null)
            {
                // 1. Получаем данные из UI
                DateTime newDate = dtpTransactionDate.Value.Date;
                string newDescription = txtTransactionDescription.Text.Trim();
                decimal amount = nudTransactionAmount.Value;
                TransactionType  type  =  rdbTransactionExpense.Checked  ?
TransactionType.Expense : TransactionType.Income;
                Category          selectedCategory          =
(Category)cmbTransactionCategory.SelectedItem;

                if          (selectedCategory          !=          null          &&
!string.IsNullOrEmpty(newDescription))
                {

```

```

        // Создаем *новую* транзакцию с обновленными значениями
        Transaction newTransaction = new Transaction(newDate,
newDescription, amount, selectedCategory, type);

        // Обновляем существующую транзакцию
        dbHelper.UpdateTransaction(newTransaction, selectedTransaction);

        accountingData.UpdateTransaction(selectedTransaction,
newTransaction); // Используем существующую транзакцию

        // Обновляем DataGridView
        transactionsBindingSource.ResetBindings(false);

        UpdateBalance();
        UpdateCategoryTotals();
        UpdateChart();
        ClearTransactionFields();
    }
    else
    {
        MessageBox.Show("Пожалуйста, выберите категорию и введите
описание.", "Предупреждение", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
    }
}
}
else
{
    MessageBox.Show("Выберите транзакцию для редактирования.",
"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
}

private void btnDeleteTransaction_Click(object sender, EventArgs e)
{
    if (dgvTransactions.SelectedRows.Count > 0)
    {
        Transaction selectedTransaction =
(Transaction)dgvTransactions.SelectedRows[0].DataBoundItem;
        if (selectedTransaction != null)
        {
            DialogResult result = MessageBox.Show($"Вы уверены, что хотите
удалить транзакцию '{selectedTransaction.Description}'?", "Подтверждение
удаления", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
            if (result == DialogResult.Yes)

```

```

        {
            // Удаляем транзакцию из базы данных
            dbHelper.DeleteTransaction(selectedTransaction.Date,
selectedTransaction.Description, selectedTransaction.Amount);
            // Удаляем транзакцию из списка в памяти
            accountingData.DeleteTransaction(selectedTransaction.Date,
selectedTransaction.Description, selectedTransaction.Amount);
            transactionsBindingSource.ResetBindings(false);
            UpdateBalance();
            UpdateCategoryTotals();
            UpdateChart();
        }
    }
}
else
{
    MessageBox.Show("Выберите транзакцию для удаления.",
"Предупреждение", MessageBoxButtons.OK, MessageBoxIcon.Warning);
}
}

```

```

private void ClearTransactionFields()
{
    txtTransactionDescription.Clear();
    nudTransactionAmount.Value = 0;
    rdbTransactionIncome.Checked = true;
    dtpTransactionDate.Value = DateTime.Now; // Добавлено: сброс даты
}

```

```

private void UpdateBalance()
{
    decimal balance = accountingData.CalculateBalance();
    lblBalance.Text = $"Баланс: {balance:C}";
}

```

```

private void UpdateCategoryTotals()
{
    Dictionary<string, decimal> categoryTotals =
accountingData.CalculateCategoryTotals();
}

```

```

// Вывод в ListBox
lstCategoryTotals.Items.Clear();
foreach (var kvp in categoryTotals)
{
    lstCategoryTotals.Items.Add($"{kvp.Key}: {kvp.Value:C}");
}

```

```

    }
}

private void UpdateChart()
{
    chtCategoryTotals.Series.Clear();

    Series series = new Series("Category Totals");
    series.ChartType = SeriesChartType.Pie;

    Dictionary<string, decimal> chartData =
accountingData.GetDataForChart();

    foreach (var kvp in chartData)
    {
        series.Points.AddXY(kvp.Key, kvp.Value);
    }

    chtCategoryTotals.Series.Add(series);
}

private void btnSaveTransactions_Click(object sender, EventArgs e)
{
    SaveDataToFile();
}

private void btnLoadTransactions_Click(object sender, EventArgs e)
{
    LoadDataFromFile();
}

// Обработчики событий для текстовых полей (можно добавлять
валидацию)
private void txtNameCategory_TextChanged(object sender, EventArgs e) { }
private void txtDescriptionCategory_TextChanged(object sender, EventArgs
e) { }
private void dtpTransactionDate_ValueChanged(object sender, EventArgs e)
{ }
private void cmbTransactionCategory_SelectedIndexChanged(object sender,
EventArgs e) { }
private void txtTransactionDescription_TextChanged(object sender,
EventArgs e) { }
private void nudTransactionAmount_ValueChanged(object sender,
EventArgs e) { }
private void rdbTransactionIncome_CheckedChanged(object sender,
EventArgs e) { }

```

```

private void rdbTransactionExpense_CheckedChanged(object sender,
EventArgs e) { }

private void btnCalculateBalance_Click(object sender, EventArgs e)
{
    UpdateBalance();
}

// Обработчик события SelectionChanged для DataGridView
private void dgvTransactions_SelectionChanged(object sender, EventArgs e)
{
    if (dgvTransactions.SelectedRows.Count > 0)
    {
        Transaction selectedTransaction =
(Transaction)dgvTransactions.SelectedRows[0].DataBoundItem;
        if (selectedTransaction != null)
        {
            // Заполняем поля ввода данными выбранной транзакции
            dtpTransactionDate.Value = selectedTransaction.Date;
            txtTransactionDescription.Text = selectedTransaction.Description;
            nudTransactionAmount.Value = selectedTransaction.Amount;
            cmbTransactionCategory.SelectedItem =
selectedTransaction.Category;

            if (selectedTransaction.Type == TransactionType.Income)
            {
                rdbTransactionIncome.Checked = true;
            }
            else
            {
                rdbTransactionExpense.Checked = true;
            }
        }
    }
}
}

```

## Содержимое файла Transaction.cs

```
using System;

namespace Книга_учета
{
    // Тип операции (доход/расход)
    public enum TransactionType
    {
        Expense,
        Income
    }

    // Класс для операции
    public class Transaction
    {
        public DateTime Date { get; set; }
        public string Description { get; set; }
        public decimal Amount { get; set; }
        public Category Category { get; set; }
        public TransactionType Type { get; set; }

        public Transaction()
        {
            // Конструктор по умолчанию необходим для десериализации JSON
            Date = DateTime.Now; // Инициализация для избежания
NullReferenceException
            Description = string.Empty; // Инициализация для избежания
NullReferenceException
            Amount = 0.0m; // Инициализация для избежания
NullReferenceException
            Category = new Category(); // Инициализация для избежания
NullReferenceException
            Type = TransactionType.Expense; // Значение по умолчанию
        }

        public Transaction(DateTime date, string description, decimal amount,
Category category, TransactionType type)
        {
            Date = date;
            Description = description;
            Amount = amount;
            Category = category;
            Type = type;
        }
    }
}
```



}  
}  
}

## Содержимое файла AccountingData.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Newtonsoft.Json;

namespace Книга_учета
{
    // Класс для хранения всех данных
    public class AccountingData
    {
        //private DatabaseHelper dbHelper; // Убрали

        public List<Category> Categories { get; set; } = new List<Category>();
        public List<Transaction> Transactions { get; set; } = new
List<Transaction>();

        public AccountingData() // Конструктор по умолчанию
        {
            Categories = new List<Category>();
            Transactions = new List<Transaction>();
        }

        // Конструктор
        /* public AccountingData(DatabaseHelper dbHelper)
        {
            this.dbHelper = dbHelper;
            LoadDataFromDatabase();
        }
        */ // удаляем вызов

        /*
        private void LoadDataFromDatabase()
        {
            Categories = dbHelper.GetAllCategories();
            Transactions = dbHelper.GetAllTransactions();
        }
        */ //удалили

        // Методы CRUD для категорий
        public void AddCategory(Category category)
        {

```

```

        //if (!Categories.Any(c => c.Name == category.Name))
        //{
        //    dbHelper.AddCategory(category);
        //    Categories.Add(category);
        //}
        //else
        //{
        //    Console.WriteLine("Категория с таким именем уже существует.");
        //}
        Categories.Add(category);
    }

    public void UpdateCategory(string oldName, Category newCategory)
    {
        Category existingCategory = Categories.FirstOrDefault(c => c.Name == oldName);
        if (existingCategory != null)
        {
            // dbHelper.UpdateCategory(newCategory);
            existingCategory.Name = newCategory.Name;
            existingCategory.Description = newCategory.Description;
        }
        else
        {
            Console.WriteLine("Категория не найдена.");
        }
    }

    public void DeleteCategory(string categoryName)
    {
        Category categoryToRemove = Categories.FirstOrDefault(c => c.Name == categoryName);
        if (categoryToRemove != null)
        {
            // dbHelper.DeleteCategory(categoryName);
            Categories.Remove(categoryToRemove);
            // Удалить все транзакции, связанные с этой категорией. Важно!
            Transactions.RemoveAll(t => t.Category.Name == categoryName);
        }
        else
        {
            Console.WriteLine("Категория не найдена.");
        }
    }
}

```

```

// Методы CRUD для операций
public void AddTransaction(Transaction transaction)
{
    // dbHelper.AddTransaction(transaction);
    Transactions.Add(transaction);
}

public void UpdateTransaction(Transaction originalTransaction, Transaction
newTransaction)
{
    // Сначала ищем существующую транзакцию по дате, описанию и
сумме
    Transaction existingTransaction = Transactions.FirstOrDefault(t =>
        t.Date == originalTransaction.Date &&
        t.Description == originalTransaction.Description &&
        t.Amount == originalTransaction.Amount);

    if (existingTransaction != null)
    {
        // Обновляем существующую транзакцию в базе данных
        // dbHelper.UpdateTransaction(newTransaction, originalTransaction);

        // Обновляем свойства существующей транзакции в памяти
        existingTransaction.Date = newTransaction.Date;
        existingTransaction.Description = newTransaction.Description;
        existingTransaction.Amount = newTransaction.Amount;
        existingTransaction.Category = newTransaction.Category;
        existingTransaction.Type = newTransaction.Type;
    }
    else
    {
        Console.WriteLine("Транзакция не найдена для обновления.");
    }
}

public void DeleteTransaction(DateTime date, string description, decimal
amount)
{
    Transactions.RemoveAll(t => t.Date == date && t.Description ==
description && t.Amount == amount);
}

// Подсчет баланса
public decimal CalculateBalance()
{

```

```

        decimal income = Transactions.Where(t => t.Type ==
TransactionType.Income).Sum(t => t.Amount);
        decimal expense = Transactions.Where(t => t.Type ==
TransactionType.Expense).Sum(t => t.Amount);
        return income - expense;
    }

    // Подсчет суммы по категориям
    public Dictionary<string, decimal> CalculateCategoryTotals()
    {
        return Transactions.GroupBy(t => t.Category.Name)
            .ToDictionary(g => g.Key, g => g.Sum(t => (t.Type ==
TransactionType.Income ? t.Amount : -t.Amount)));
    }

    // Подготовка данных для графика (пример)
    public Dictionary<string, decimal> GetDataForChart()
    {
        return CalculateCategoryTotals(); // Используем уже посчитанные
суммы по категориям
    }
}

```

## Содержимое файла DatabaseHelper.cs

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.IO;
using System.Linq;

namespace Книга_учета
{
    public class DatabaseHelper : IDisposable
    {
        private string dbFilePath;
        private string connectionString;
        public SQLiteConnection connection { get; private set; } // getter public

        public DatabaseHelper(string dbFilePath)
        {
            this.dbFilePath = dbFilePath;
            this.connectionString = $"Data Source={dbFilePath};Version=3;";
            connection = new SQLiteConnection(connectionString);
        }

        public void CreateDatabaseIfNotExists()
        {
            if (!File.Exists(dbFilePath))
            {
                SQLiteConnection.CreateFile(dbFilePath);
                CreateTables();
            }
        }

        private void CreateTables()
        {
            using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
            {
                connection.Open();

                // Создание таблицы Categories
                string createCategoriesTableQuery = @"
                CREATE TABLE Categories (
```

```

        Name TEXT PRIMARY KEY,
        Description TEXT
    );";
    SQLiteCommand createCategoriesCommand = new
SQLiteCommand(createCategoriesTableQuery, connection);
    createCategoriesCommand.ExecuteNonQuery();

    // Создание таблицы Transactions
    string createTransactionsTableQuery = @"
        CREATE TABLE Transactions (
            Date TEXT,
            Description TEXT,
            Amount REAL,
            CategoryName TEXT,
            Type INTEGER,
            FOREIGN KEY (CategoryName) REFERENCES
Categories(Name)
        );";
    SQLiteCommand createTransactionsCommand = new
SQLiteCommand(createTransactionsTableQuery, connection);
    createTransactionsCommand.ExecuteNonQuery();
    }
}

// Методы для работы с категориями
public void AddCategory(Category category)
{
    using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
    {
        connection.Open();
        string insertQuery = "INSERT INTO Categories (Name, Description)
VALUES (@Name, @Description)";
        SQLiteCommand insertCommand = new SQLiteCommand(insertQuery,
connection);
        insertCommand.Parameters.AddWithValue("@Name", category.Name);
        insertCommand.Parameters.AddWithValue("@Description",
category.Description);
        insertCommand.ExecuteNonQuery();
    }
}

public void UpdateCategory(Category category)
{

```

```

        using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
        {
            connection.Open();
            string updateQuery = "UPDATE Categories SET Description =
@Description WHERE Name = @Name;";
            SQLiteCommand updateCommand = new
SQLiteCommand(updateQuery, connection);
            updateCommand.Parameters.AddWithValue("@Name",
category.Name);
            updateCommand.Parameters.AddWithValue("@Description",
category.Description);
            updateCommand.ExecuteNonQuery();
        }
    }

    public void DeleteCategory(string categoryName)
    {
        using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
        {
            connection.Open();

            // Сначала удаляем транзакции, связанные с этой категорией
            string deleteTransactionsQuery = "DELETE FROM Transactions
WHERE CategoryName = @CategoryName;";
            SQLiteCommand deleteTransactionsCommand = new
SQLiteCommand(deleteTransactionsQuery, connection);

            deleteTransactionsCommand.Parameters.AddWithValue("@CategoryName",
categoryName);
            deleteTransactionsCommand.ExecuteNonQuery();

            // Затем удаляем саму категорию
            string deleteCategoryQuery = "DELETE FROM Categories WHERE
Name = @Name;";
            SQLiteCommand deleteCategoryCommand = new
SQLiteCommand(deleteCategoryQuery, connection);
            deleteCategoryCommand.Parameters.AddWithValue("@Name",
categoryName);
            deleteCategoryCommand.ExecuteNonQuery();
        }
    }

    public List<Category> GetAllCategories()

```



```

    {
        List<Category> categories = new List<Category>();
        using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
        {
            connection.Open();
            string selectQuery = "SELECT Name, Description FROM Categories;";
            SQLiteCommand selectCommand = new SQLiteCommand(selectQuery,
connection);
            using (SQLiteDataReader reader = selectCommand.ExecuteReader())
            {
                while (reader.Read())
                {
                    Category category = new Category(reader.GetString(0),
reader.GetString(1));
                    categories.Add(category);
                }
            }
        }
        return categories;
    }

```

```

// Методы для работы с транзакциями
public void AddTransaction(Transaction transaction)
{
    using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
    {
        connection.Open();
        string insertQuery = "INSERT INTO Transactions (Date, Description,
Amount, CategoryName, Type) VALUES (@Date, @Description, @Amount,
@CategoryName, @Type);";
        SQLiteCommand insertCommand = new SQLiteCommand(insertQuery,
connection);
        insertCommand.Parameters.AddWithValue("@Date",
transaction.Date.ToString("yyyy-MM-dd HH:mm:ss"));
        insertCommand.Parameters.AddWithValue("@Description",
transaction.Description);
        insertCommand.Parameters.AddWithValue("@Amount",
transaction.Amount);
        insertCommand.Parameters.AddWithValue("@CategoryName",
transaction.Category.Name);
        insertCommand.Parameters.AddWithValue("@Type",
(int)transaction.Type);
        insertCommand.ExecuteNonQuery();
    }
}

```

```
    }  
}
```

```
public void UpdateTransaction(Transaction newTransaction, Transaction  
originalTransaction)
```

```
{  
    using (SQLiteConnection connection = new  
SQLiteConnection(connectionString))  
    {  
        connection.Open();  
        string updateQuery = @"  
UPDATE Transactions  
SET Date = @Date,  
    Description = @Description,  
    Amount = @Amount,  
    CategoryName = @CategoryName,  
    Type = @Type  
WHERE Date = @OriginalDate  
    AND Description = @OriginalDescription  
    AND Amount = @OriginalAmount;";  
  
        SQLiteCommand updateCommand = new  
SQLiteCommand(updateQuery, connection);  
        updateCommand.Parameters.AddWithValue("@Date",  
newTransaction.Date.ToString("yyyy-MM-dd HH:mm:ss"));  
        updateCommand.Parameters.AddWithValue("@Description",  
newTransaction.Description);  
        updateCommand.Parameters.AddWithValue("@Amount",  
newTransaction.Amount);  
        updateCommand.Parameters.AddWithValue("@CategoryName",  
newTransaction.Category.Name);  
        updateCommand.Parameters.AddWithValue("@Type",  
(int)newTransaction.Type);  
        updateCommand.Parameters.AddWithValue("@OriginalDate",  
originalTransaction.Date.ToString("yyyy-MM-dd HH:mm:ss"));  
        updateCommand.Parameters.AddWithValue("@OriginalDescription",  
originalTransaction.Description);  
        updateCommand.Parameters.AddWithValue("@OriginalAmount",  
originalTransaction.Amount);  
        updateCommand.ExecuteNonQuery();  
    }  
}
```

```
public void DeleteTransaction(DateTime date, string description, decimal  
amount)
```

```

    {
        using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
        {
            connection.Open();
            string deleteQuery = "DELETE FROM Transactions WHERE Date =
@Date AND Description = @Description AND Amount = @Amount;";
            SQLiteCommand deleteCommand = new SQLiteCommand(deleteQuery,
connection);
            deleteCommand.Parameters.AddWithValue("@Date",
date.ToString("yyyy-MM-dd HH:mm:ss"));
            deleteCommand.Parameters.AddWithValue("@Description",
description);
            deleteCommand.Parameters.AddWithValue("@Amount", amount);
            deleteCommand.ExecuteNonQuery();
        }
    }

    public List<Transaction> GetAllTransactions()
    {
        List<Transaction> transactions = new List<Transaction>();
        using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
        {
            connection.Open();
            string selectQuery = "SELECT Date, Description, Amount,
CategoryName, Type FROM Transactions;";
            SQLiteCommand selectCommand = new SQLiteCommand(selectQuery,
connection);
            using (SQLiteDataReader reader = selectCommand.ExecuteReader())
            {
                while (reader.Read())
                {
                    DateTime date = DateTime.Parse(reader.GetString(0));
                    string description = reader.GetString(1);
                    decimal amount = Convert.ToDecimal(reader.GetValue(2));
                    string categoryName = reader.GetString(3);
                    TransactionType type =
(TransactionType)Convert.ToInt32(reader.GetValue(4));

                    // Получаем категорию по имени
                    Category category = GetAllCategories().FirstOrDefault(c =>
c.Name == categoryName);

```

```

        Transaction transaction = new Transaction(date, description,
amount, category, type);
        transactions.Add(transaction);
    }
}
}
return transactions;
}
public void ClearAllData()
{
    using (SQLiteConnection connection = new
SQLiteConnection(connectionString))
    {
        connection.Open();

        // Удаляем все данные из таблицы Transactions
        string deleteTransactionsQuery = "DELETE FROM Transactions;";
        SQLiteCommand deleteTransactionsCommand = new
SQLiteCommand(deleteTransactionsQuery, connection);
        deleteTransactionsCommand.ExecuteNonQuery();

        // Удаляем все данные из таблицы Categories
        string deleteCategoriesQuery = "DELETE FROM Categories;";
        SQLiteCommand deleteCategoriesCommand = new
SQLiteCommand(deleteCategoriesQuery, connection);
        deleteCategoriesCommand.ExecuteNonQuery();
    }
}

// Implementation of IDisposable
private bool disposed = false;

protected virtual void Dispose(bool disposing)
{
    if (!disposed)
    {
        if (disposing)
        {
            // Dispose managed resources
            if (connection != null)
            {
                connection.Close();
                connection.Dispose();
                connection = null;
            }
        }
    }
}

```

```
    }  
    // Dispose unmanaged resources  
  
    disposed = true;  
    }  
}  
  
public void Dispose()  
{  
    Dispose(true);  
    GC.SuppressFinalize(this);  
}  
  
~DatabaseHelper()  
{  
    Dispose(false);  
}  
}  
}
```