

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc136615797)

[1. Описание информационной модели предметной области 4](#_Toc136615798)

[2. Разработка ER-диаграммы 6](#_Toc136615799)

[3. Программирование объектов базы данных 10](#_Toc136615800)

[4. Разработка интерфейса пользователя 13](#_Toc136615801)

[ЗАКЛЮЧЕНИЕ 25](#_Toc136615802)

[ССЫЛКИ НА ИСТОЧНИКИ 26](#_Toc136615803)

[ПРИЛОЖЕНИЕ А 27](#_Toc136615804)

[ПРИЛОЖЕНИЕ Б 54](#_Toc136615805)

# ВВЕДЕНИЕ

В рамках данной темы необходимо построить базу данных, которая помогла бы производить учёт информации, которая относится к работе центра, занимающегося продажей автомобилей. Данная тема является актуальной, так как спроектированная надлежащим образом база данных дала бы следующие важные возможности:

* Учёт информации по каталогу товаров;
* Учёт информации по списку клиентов;
* Учёт информации по списку поставщиков;
* Учёт информации по списку клиентов;
* Учёт информации по продаже того или иного товара конкретному клиенту;
* Учёт информации по поставке того или иного товара конкретным поставщиком.

Также благодаря возможностям базы данных появляется возможность наиболее лёгким, безопасным и надёжным способом провести добавление, изменение, удаление информации, а также просмотр информации по конкретным параметрам и внятных отчётов.

Чтобы достичь выполнения данной цели, необходимо выполнить следующие задачи:

* Описать информационную модель предметной области;
* Разработать ER-диаграмму;
* Запрограммировать объекты базы данных;
* Разработать интерфейс пользователя.

# Описание информационной модели предметной области

Поскольку центр по продаже автомобилей – это офисы, совершающие акт продажи тех или иных автомобилей, то естественным образом возникает необходимость в определённой структуре, состоящей из взаимосвязанных элементов. Для объяснения этой структуры необходимо привести информацию об элементах базы данных.

Поскольку в рамках данной работы было решено взять за работу не один конкретный филиал, а целую сеть центров по продаже автомобилей, то из этого вытекает необходимость в информации о каждом из филиалов, для чего необходима соответствующая таблица под названием «Филиалы» со следующими данными: город, адрес, телефон.

Поскольку каждому центру необходимо закупаться автомобилями для продажи, то рациональным ходом было бы также иметь и для этого момента таблицу «Поставщики» со следующими данными: наименование поставщика, город и улица, из которых будет совершаться поставка, телефон и рейтинг.

Как и любое предприятие, необходимо иметь информацию о клиентах, совершивших или совершающих покупку автомобиля, в виде таблицы «Клиенты», в которую включить можно данные о ФИО, городе и адресе проживания и телефоне.

И, естественно, необходима информацию о самих товарах, для чего понадобится таблица «Товары», а среди данных о товаре должны быть наименование, цвет, дата изготовления и текущая цена. Но когда речь заходит об автопромышленности, то всегда встаёт необходимость учесть наличие у разных фирм своих конкретных моделей автомобилей, из чего вытекает создание таблицы «Наименования\_Товаров», в которой будет перечисление фирм с имеющимися конкретно у них определённых моделей автомобилей

Как и любому предприятию, необходима информацию о текущих сотрудниках, а также информации о текущей необходимом списке должностей. В информацию о должностях для таблицы «Должности »достаточно иметь название должности и оклад, а о сотруднике – ФИО, дате рождения, поле, телефоне, паспортных данных, ИИН и занимаемой должности.

Также важным является полезным информация о текущем ассортименте товаров у филиалов, для чего потребуется соответствующая таблица «Ассортимент\_Товаров», в которой будет информации о наличии в данном филиале того или иного товара.

Теперь, если дело доходит дело до сделки на продажу, то необходимым было бы иметь информацию о том, какой товар был продан, в каком филиале, какому клиенту, когда были подписан договор о продаже и выполнена доставка, какой менеджер был ответственен за данную продажу, а также стоимость данной продажи, равная текущей цене продаваемого товара, в связи с чем пригодится таблица «Продажи». А в сценарии с поставкой - какой товар был поставлен, в какой филиал, каким поставщиком, когда были подписан договор о поставке и выполнена доставка, какой менеджер был ответственен за данную продажу, а также стоимость данной поставки, которая будет храниться в таблице «Поставки».

Помимо информации, описанной выше, необходимым шагом было бы иметь возможность статистику, в которой бы приводилось количество имеющихся товаров на филиал, максимальное и минимальное количество проданных или поставленных товаров на филиал по датам (год и месяц) и максимальная и минимальная сумма заработка от продаж или трат на поставку по датам (год и месяц), которую можно поместить в раздел «Отчёты». Плюс возможность как фильтровать необходимую для отображения информацию с помощью «Фильтр», так и отменить саму фильтрацию, тем самым показав полное содержание таблицы.

# Разработка ER-диаграммы

Для построения базы данных необходимо определиться с её инфологической моделью, для чего необходимо перечислить необходимые сущности, их атрибуты, а также связи между ними.

В качестве первого шага было бы лучше кратко описать задачи создаваемой базы данных:

* Хранение информации о клиентах;
* Хранение информации о филиалах;
* Хранение информации о поставщиках;
* Хранение информации о фирмах и имеющихся у них моделей автомобилей;
* Хранение информации о товарах, то есть экземплярах автомобилей;
* Хранение информации о должностях;
* Хранение информации о сотрудниках;
* Следить за наличием экземпляра того или иного товара в филиалах;
* Печатать накладные на проданные товары;
* Печатать накладные на поставленные товары.

Теперь можно перейти к обобщённому списку необходимых для работы сущностей с имеющимися у них атрибутами и связями с другими сущностями:

* Клиенты: Номер клиента (первичный ключ, имеющий тип serial), ФИО (тип varchar), Город (тип varchar), Адрес (тип varchar), Телефон (тип varchar);
* Филиалы: Номер филиала (первичный ключ, имеющий тип serial), Город (тип varchar), Адрес (тип varchar), Телефон (тип varchar);
* Поставщики: Номер\_поставщика (первичный ключ, имеющий тип serial), Наименование (тип varchar), Город (тип varchar), Адрес (тип varchar), Телефон (тип varchar), Рейтинг (тип varchar);
* Наименования\_Товаров: Код\_имени (первичный ключ, имеющий тип serial), Фирма (тип varchar), Модель (тип varchar);
* Товары: Код\_товара (первичный ключ типа varchar), Наименование (вторичный ключ типа serial, ссылающийся на первичный ключ «Код\_имени» сущности «Наименования\_Товаров»), Цвет (тип varchar), Изготовлено (тип date), Текущая\_цена (типа numeric с 0 в качестве значения по умолчанию);
* Должности: Номер\_должности (первичный ключ типа serial), Название\_должности (тип varchar), Оклад (тип numeric с 0 в качестве значения по умолчанию);
* Сотрудники: Табельный\_номер (первичный ключ типа varchar), ФИО (тип varchar), Дата\_рождения (тип date), Пол (тип varchar с количеством символов, равным 1), Телефон (тип varchar), Паспортные\_данные (тип varchar), ИНН (тип varchar с количеством символов, равным 12), Должность (вторичный ключ типа serial, ссылающийся на первичный ключ «Номер\_должности» сущности «Должности»);
* Ассортимент Филиалов: Филиал (вторичный ключ типа serial, ссылающийся на первичный ключ «Номер\_филиала» сущности «Филиалы»), Товар (вторичный ключ типа varchar, ссылающийся на первичный ключ «Код\_товара» сущности «Товары»), Наличие (тип bool);
* Продажи: Номер\_продажи (первичный ключ типа serial), Товар (вторичный ключ типа varchar, ссылающийся на первичный ключ «Код\_товара» сущности «Товары»), Филиал (вторичный ключ типа serial, ссылающийся на первичный ключ «Номер\_филиала» сущности «Филиалы»), Клиент (вторичный ключ типа serial, ссылающийся на первичный ключ «Номер\_клиента» сущности «Клиенты»), Подписано (тип date), Доставлено (тип date с ограничением, что Доставлено >= Подписано), Менеджер (вторичный ключ типа varchar, ссылающийся на первичный ключ «Табельный\_номер» сущности «Сотрудники»), Стоимость (тип numeric с 0 в качестве значения по умолчанию);
* Поставки: Номер\_поставки (первичный ключ типа serial), Товар (вторичный ключ типа varchar, ссылающийся на первичный ключ «Код\_товара» сущности «Товары»), Филиал (вторичный ключ типа serial, ссылающийся на первичный ключ «Номер\_филиала» сущности «Филиалы»), Поставщик (вторичный ключ типа serial, ссылающийся на первичный ключ «Номер\_поставщика» сущности «Поставщики»), Подписано (тип date), Доставлено (тип date с ограничением, что Доставлено >= Подписано), Менеджер (вторичный ключ типа varchar, ссылающийся на первичный ключ «Табельный\_номер» сущности «Сотрудники»), Стоимость (тип numeric с 0 в качестве значения по умолчанию).

Стоит отметить, что в сущности комбинация атрибутов «Филиал» и «Товар» должна быть уникальной, так как в одном и том же филиале не может быть одного и того же экземпляра автомобиля. Плюс в таблицах «Продажи» и «Поставки» было бы лучше у всех атрибутов, являющихся внешними ключами, убрать ограничение, запрещающее значение «NULL», в рамках данных сущностей удобней было бы удалённые значения заменить на альтернативные, вместо полного удаления строки.

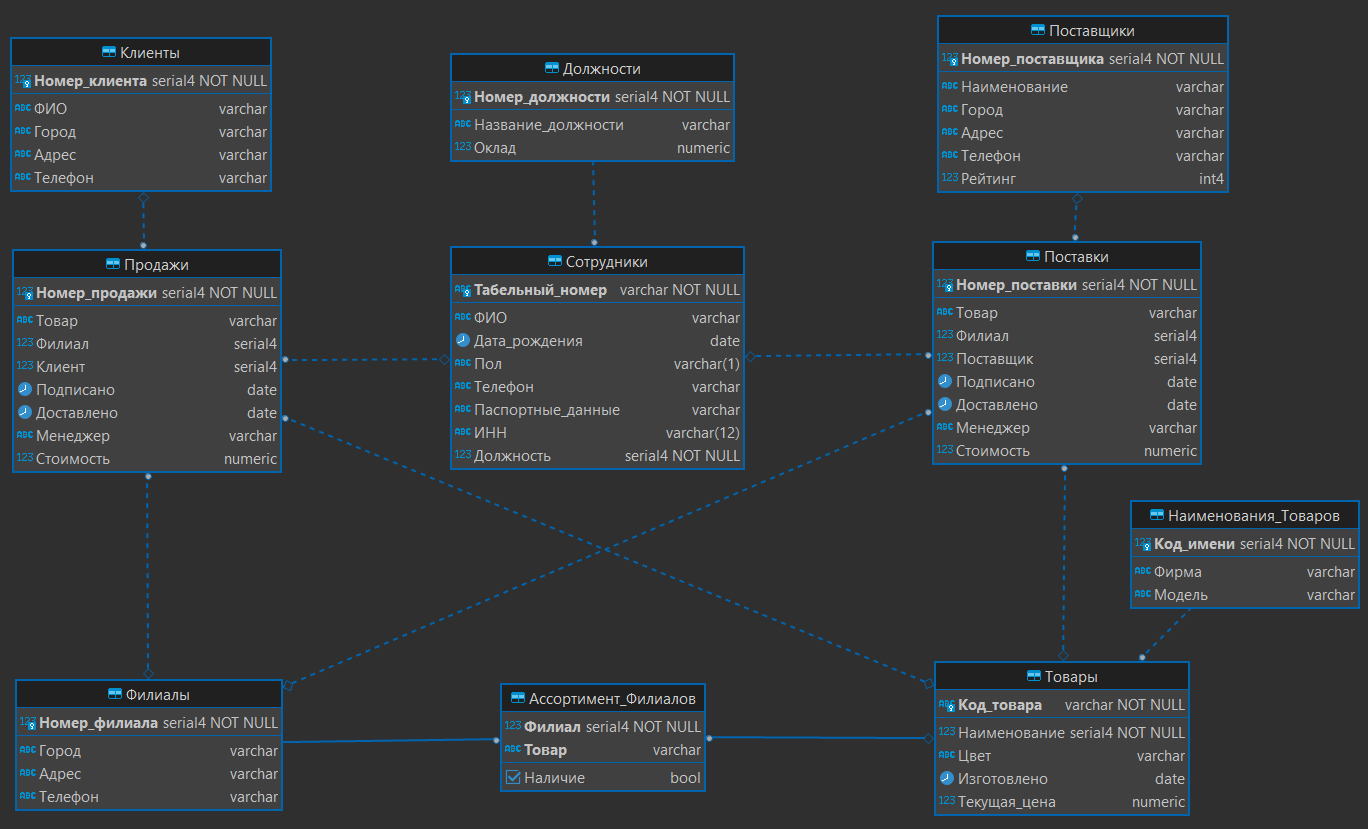


Рисунок 1. ER-диаграмма

Если отдельно затрагивать вопрос связи сущностей, то необходимо вспомнить, что связи «Много-ко-многим» не допускается, и нужно с помощью промежуточных сущностей сделать со всеми связь «Один-ко-многим». Поэтому между сущностями, содержащими информацию о продаваемых товарах, клиентах, филиале и ответственном сотруднике, сделать ту, которая бы объединяла эти данные в одной отдельной таблице. То же самое касается и сущностей, содержащих информацию о поставляемых товарах, поставщиках, филиале и ответственном сотруднике. А ссылка сущностями, содержащими информацию о товарах и сотрудниках, связана с третьей нормальной формы, а не с правилами связи ER-диаграмм.

Также стоит обосновать то, почему проектируемая база данных находится в третьей нормальной форме. Данная база данных находится в первой нормальной форме, так как каждой сущности соответствует своя отдельная таблица. Помимо этого, база данных находится во второй нормальной форме, так как в таблице «Ассортимент\_Филиалов» атрибут «Наличие» не зависит от части сложного ключа, в а в остальных сложные ключи отсутствуют. А также база данных находится в третьей нормальной форме, потому что во всех таблицах все неключевые атрибуты являются взаимно независимыми, так, например, полное наименование товара, содержащее названия фирмы и модели, выделено в отдельное отношение.

# Программирование объектов базы данных

В качестве основной СУБД было решено выбрать PostgreSQL, так как, согласно [3], среди преимуществ имеются:

* Полная SQL-совместимость;
* Сообщество: Поддержка опытным сообществом;
* Объектно-ориентированность;
* Поддержка целостности данных;
* Поддержка сложных процедур;
* Наличие возможности для интеграция.

Создание требуемой базы данных нужно начать с создания схемы с помощью команды «CREATE SCHEMA»:

CREATE SCHEMA Авто;

После этого можно начать создавать таблицы с помощью команды «CREATE TABLE», например, для таблицы «Продажи»:

CREATE TABLE Авто.Продажи(

Номер\_продажи serial PRIMARY KEY,

Товар varchar REFERENCES Авто.Товары(Код\_товара),

Филиал serial REFERENCES Авто.Филиалы(Номер\_филиала),

Клиент serial REFERENCES Авто.Клиенты(Номер\_клиента),

Подписано date,

Доставлено date CHECK (Доставлено >= Подписано),

Менеджер varchar REFERENCES Авто.Сотрудники(Табельный\_номер),

Стоимость numeric DEFAULT 0

);

А для таблицы «Ассортимент Филиалов», в которой должна соблюдаться уникальность комбинации кода товара и номера филиала, код будет выглядеть следующим образом:

CREATE TABLE Авто.Ассортимент\_Филиалов(

Филиал serial REFERENCES Авто.Филиалы(Номер\_филиала),

Товар varchar REFERENCES Авто.Товары(Код\_товара),

Наличие bool,

UNIQUE(Филиал, Товар)

);

Так как в таблицах «Продажи» и «Поставки» нужно убрать ограничение на значение «NULL» для атрибутов, являющимися внешними ключами, нужно использовать команду «ALTER TABLE»:

ALTER TABLE Авто.Продажи ALTER COLUMN Филиал DROP NOT NULL;

ALTER TABLE Авто.Продажи ALTER COLUMN Клиент DROP NOT NULL;

Также стоит привести пример создания триггера с соответствующей триггерной функцией. В качестве примера можно взять тот, который работает перед удаление данных в таблице «Сотрудники»:

CREATE OR REPLACE FUNCTION Авто.update\_delete\_workers\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(Номер\_продажи) FROM Авто.Продажи WHERE Продажи.Менеджер=OLD.Табельный\_номер)<>0 THEN

UPDATE Авто.Продажи

SET Менеджер=NULL

WHERE Продажи.Менеджер=OLD.Табельный\_номер;

END IF;

IF (SELECT COUNT(Номер\_поставки) FROM Авто.Поставки WHERE Поставки.Менеджер=OLD.Табельный\_номер)<>0 THEN

UPDATE Авто.Поставки

SET Менеджер=NULL

WHERE Поставки.Менеджер=OLD.Табельный\_номер;

END IF;

IF TG\_OP = 'UPDATE' THEN

RETURN NEW;

ELSE

RETURN OLD;

END IF;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER update\_delete\_workers\_trigger

BEFORE UPDATE OR DELETE ON Авто.Сотрудники

FOR EACH ROW

EXECUTE FUNCTION Авто.update\_delete\_workers\_function();

Базовый запрос, который выполняет добавление данных в таблицу, можно рассмотреть на примере таблицы «Ассортимент Филиалов»:

INSERT INTO Авто.Ассортимент\_Филиалов (Филиал, Товар, Наличие) VALUES (1, '112', '0');

Базовый запрос, который выполняет изменение данных в таблице, выглядит следующим образом:

UPDATE {Название схемы}.{Название таблицы} SET {Название атрибута}={Новое значение}, {Набор {Название атрибута}={Новое значение}}

Базовый запрос, который выполняет удаление данных из таблицы, выглядит следующим образом:

DELETE FROM {Название схемы}.{Название таблицы}

Базовый запрос, который выполняет отображение данных в таблице, выглядит следующим образом:

SELECT \* FROM {Название схемы}.{Название таблицы}

В качестве сложного запроса, используемого для создания отчёта, можно привести тот, который показывает количество имеющихся товаров по филиалам для таблицы «Ассортимент Филиалов»:

SELECT Филиал, COUNT(Товар) AS Количество

FROM Авто.Ассортимент\_Филиалов

WHERE Наличие=true

GROUP BY Филиал

ORDER BY Филиал

Полный набор SQL-запросов для создания базы данных представлен в ПРИЛОЖЕНИИ Б.

# Разработка интерфейса пользователя

В качестве основного языка программирования необходимого интерфейса было решено использовать язык Python из-за его относительной простоты и возможности создавать проекты со сложной архитектурой.

Стоит отметить, что оформление окон для каждой из таблиц унифицировано благодаря созданию своих шаблонов для построения необходимых для взаимодействия окон каждой из таблиц, проверки качества ввода и работы с базой данных в не зависимости от конкретной таблицы. А также для лучшего взаимодействия пользователя все окна сделаны модальными, то есть пока дочерние окна открыты, исчезает возможность одновременно выполнять какие-либо действия с родительскими для них окнами.

Интерфейс приложения должен состоять из следующих окон:

* Окна выбора таблица;
* Окна выбранной таблицы;
* Окна для редактирования данных и фильтрации в выбранной таблице;
* Окна выбора отчётов;
* Окно выбранного отчёта.

В окне первого уровня, то есть в самом первом, как можно понять из описания, можно выбрать ту таблицу, с которой пользователю необходимо работать. Выбор выполняется нажатием на кнопку с соответствующим наименованием таблицы. Помимо этого, необходимым было добавить кнопку для выхода из этого окна, что означает завершение работы приложения. Также полезным было бы добавить текст, который кратко объясняет, что нужно делать в данном окне. В итоге первое окно имеет один текст и 11 кнопок.

В окне второго уровня имеется возможность не только смотреть данные, содержащиеся в таблицах, но и редактировать данные таблиц и проводить фильтрацию, в том числе отменить действие фильтрации, а для некоторых таблиц, как «Ассортимент\_Товаров», «Продажи» и «Поставки», можно переходить в окно для выбора просмотра конкретного отчёта. Возможность просмотра всех имеющихся данных оформлено в виде таблицы с заголовком, содержащим наименования атрибутов. В свойства подобных таблиц включена возможность прокручивать содержание таблицы, что является важным из-за вероятности работы с большим набором данных. Функционал остальных возможностей реализуется путём нажатия на соответствующие им кнопки. Причём те, что отвечают за добавление, изменение и удаление данных выделены в одну рамку, а другие, что отвечают за фильтрацию и просмотр отчётов, – в другую. Во всех них, кроме той, что отменяет действие фильтрации, результатом нажатия является открытие другого окна поверх другого. Помимо всего вышеперечисленного, рациональным шагом было бы сделать кнопку, не выделенную в отдельную рамку, которая бы закрывала данное окно и возвращало на предыдущее. В итоге во втором окне содержатся 1 таблица и 6-7 кнопок.

При переходе к окну добавления данных в таблицу создаётся рамка с набором полей текста и ввода для них, которые не являются первичными ключами типа serial, а также атрибутом «Стоимость» таблицы «Продажи», так как в данный атрибут должно вставляться значение атрибута «Текущая стоимость» таблицы «Товары», а также кнопки «Добавить», осуществляющую добавление введённых значений и «Назад к таблице», которая возвращает обратно ко второму окну.

При переходе к окну удаления данных из таблицы создаётся рамка с набором полей текста и ввода для них со всеми имеющимися атрибутами, а также кнопки «Удалить определённую строку», которая удаляет строку, содержащее введённое значений в соответствующем атрибуте, «Удалить всё», совершающее удаление всех строк, и «Назад к таблице».

При переходе к окну изменения данных в таблице происходит создание рамок из 2 окон третьего уровня, среди которых одна отвечает за то, в каких строках делать изменения, а другая за использование введённых значений для вставки их в таблицу с практически такими же методами проверки ввода, за исключением того, что можно не делать ввод в первую рамку, из-за чего изменение коснётся всех строк. Помимо этого, к этому окну добавляются кнопки «Изменить», отвечающей за выполнение операции изменения, и «Назад к таблице».

В окне для добавления данных необходимо заполнить все требуемые поля ввода. В случае, если ввод полностью или вообще не выполнен, то выводится соответствующее сообщение об этом при нажатии на кнопку «Добавить». Также имеется проверка ввода, то есть при вводе текста выше необходимого предела, при ссылке в поле для атрибута, являющегося внешним ключом, значения, которого нет в ссылаемой таблице, при вводе лишних символов и неверном вводе данных при нажатии на кнопку выводится сообщение в неверном вводе со списком атрибутов, в которых ввод выполнен неверно. При вводе дат, которые нарушают ограничение, в котором дата для атрибута «Подписано» должна быть меньше или равна дате для атрибута «Доставлено» в таблицах «Продажи» и «Поставки», происходит выход из окна с выводом сообщения об неверном вводе данных. А также при нарушении уникальности комбинации кода товара и номера филиалы в таблице «Ассортимент Филиалов», выводится сообщение об этой ошибке.

В случае окна для удаления данных необходимо заполнить хотя бы 1 поле ввода и нажать на кнопку «Удалить определённую строку» или просто нажать на кнопку «Удалить всё», чтобы удалялись все имеющиеся строки. В случае, если ввод вообще не был выполнен и была нажата первая кнопка, то выводится сообщение, как при добавлении данных в случае неполного ввода. Проверка на корректность ввода значений таким же образом, как в окне для добавления, хотя в случае проверки для данного окна предусмотрена отдельная функция.

В случае окна для изменения данных из-за добавления двух рамок из двух описанных выше окон третьего уровня, то при нажатии на копку «Изменить» требования на ввод данных тот же, за исключением того, что в первую рамку можно ничего не вводить. Остальные критерии проверки работают так же подобно тем окнам.

Теперь если касаться окна третьего уровня, отвечающего за фильтрацию, то внешний вид рамку повторяет тот, что у окна для удаления данных, но вместо двух кнопок имеется одна с названием «Отфильтровать» и другая, отвечающая за закрытие текущего окна. Первая кнопка, как можно понять, реализовывает отображение строк с конкретными значениями в атрибутах. Проверка на ввод и корректность ввода повторяет окно удаления данных.

В окне третьего уровня для просмотра отчётов должны быть текст, объясняющий суть данного окна, кнопки, отвечающие за открытие окон самих отчётов, а также кнопка «Назад» в самом внизу, чтобы закрыть текущее окно. Из-за специфики самих таблиц, набор отчётов у таблицы «Ассортимент Филиалов», в то время как у остальных оно одинаковое.

В окне четвёртого уровня отображения отчётов оформлено в виде таблицы, как в окнах второго уровня, а для возврата к окну предыдущего уровня должна быть реализована кнопка «Назад».

Код интерфейса представлен в ПРИЛОЖЕНИИ А.

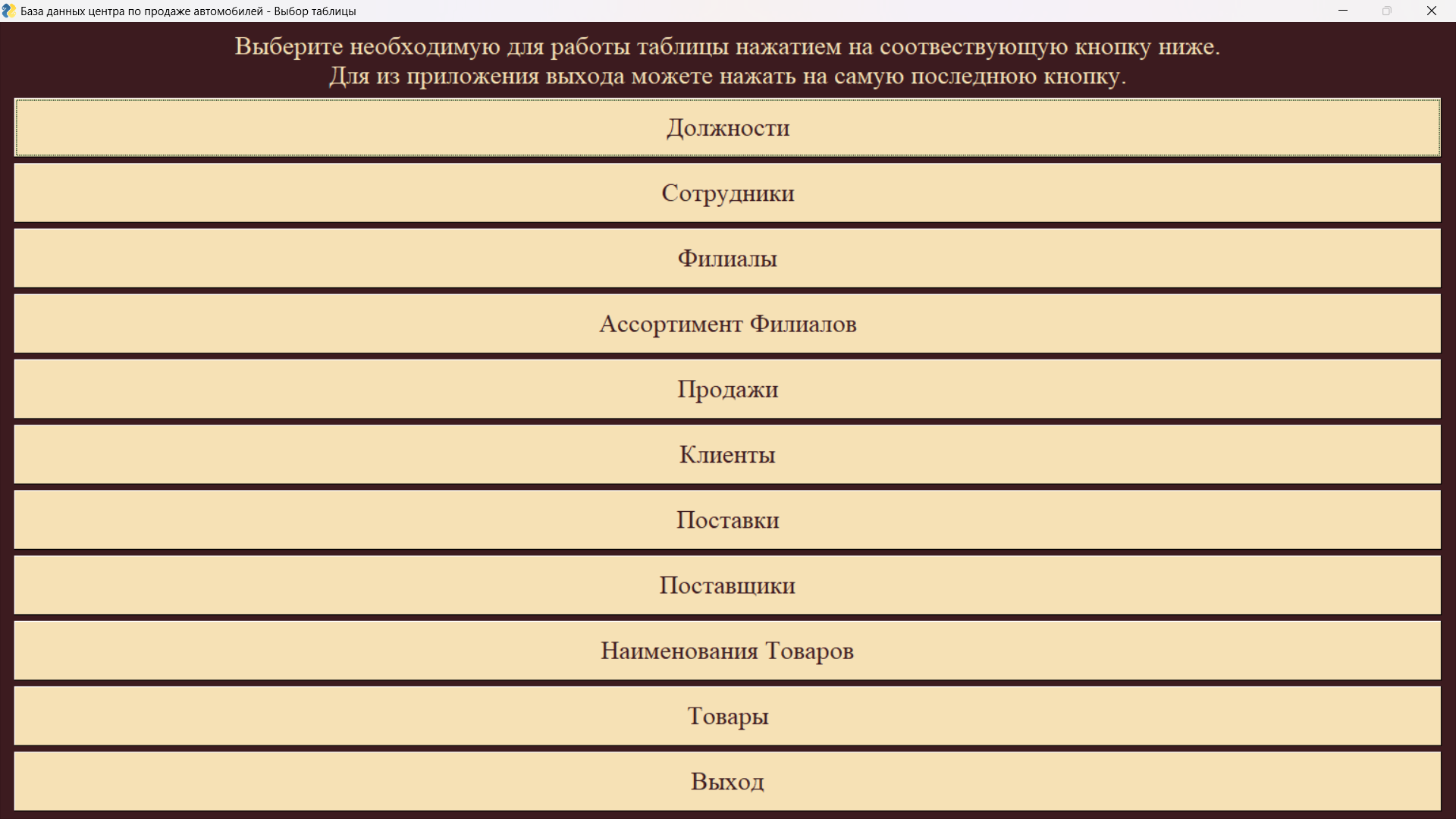


Рисунок 2. Первое окно выбора таблиц

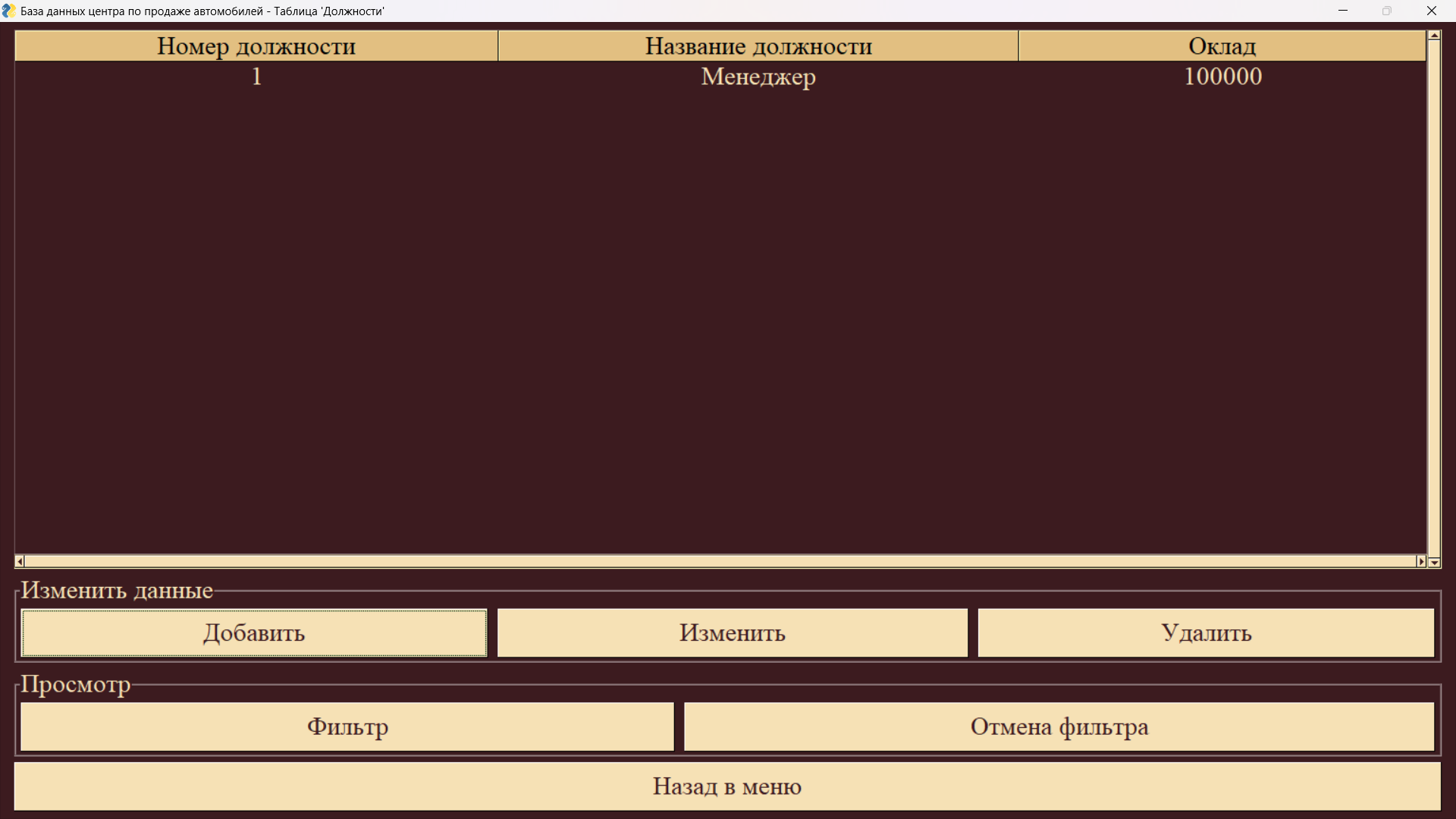


Рисунок 3. Второе окно для таблицы «Должности»

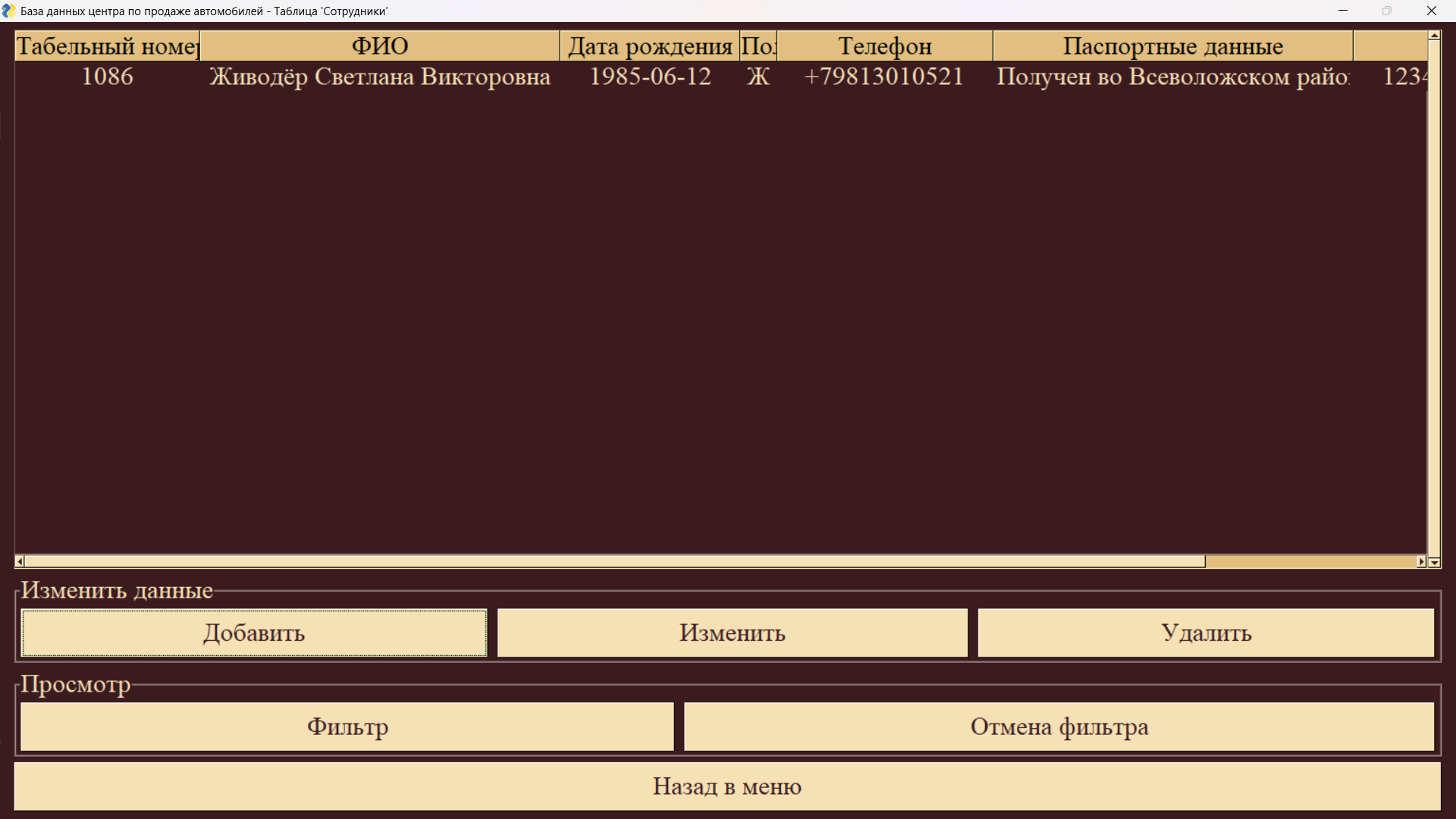


Рисунок 4. Второе окно для таблицы «Сотрудники»



Рисунок 5. Второе окно для таблицы «Филиалы»

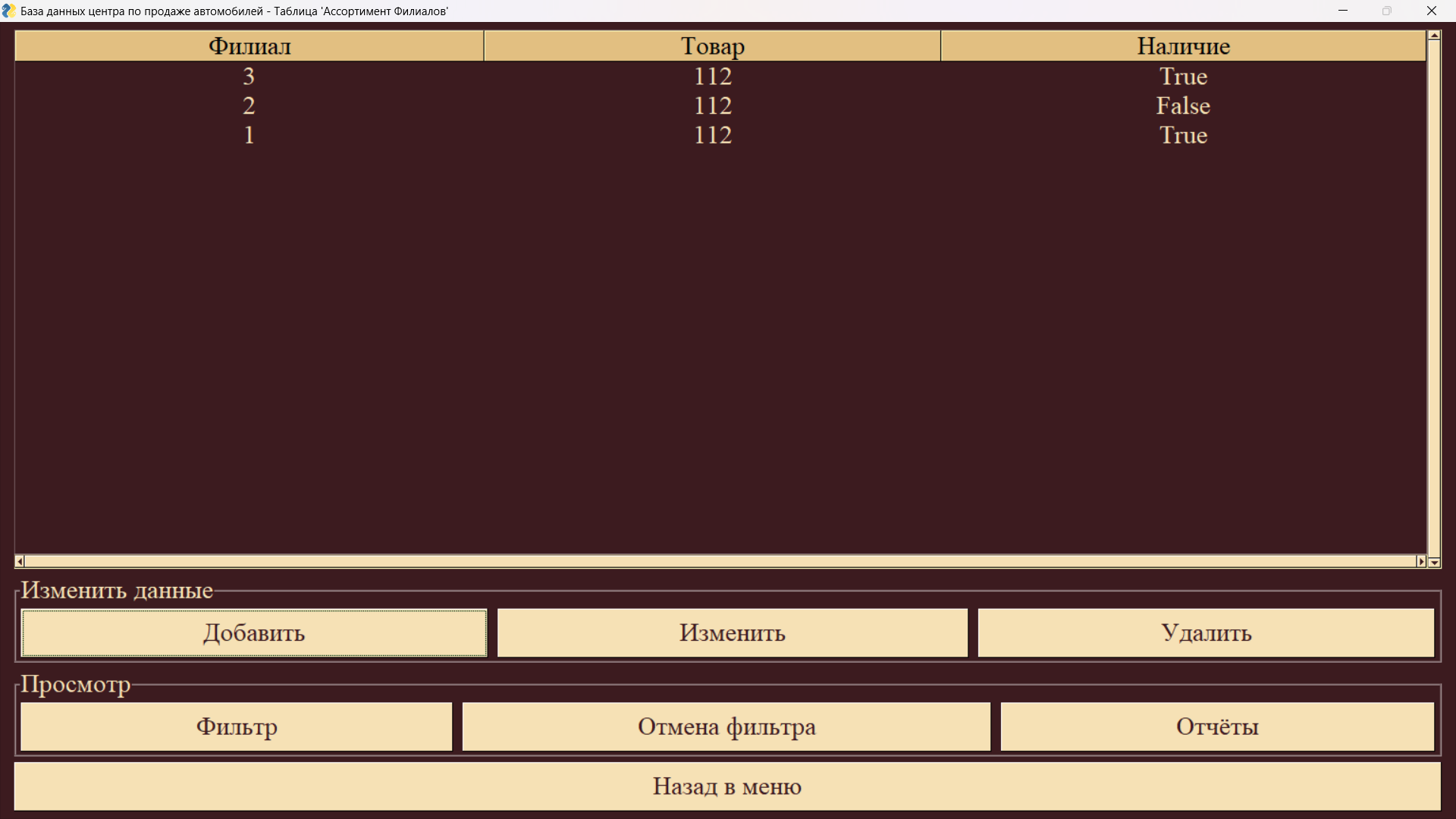


Рисунок 6. Второе окно для таблицы «Ассортимент Филиалов»

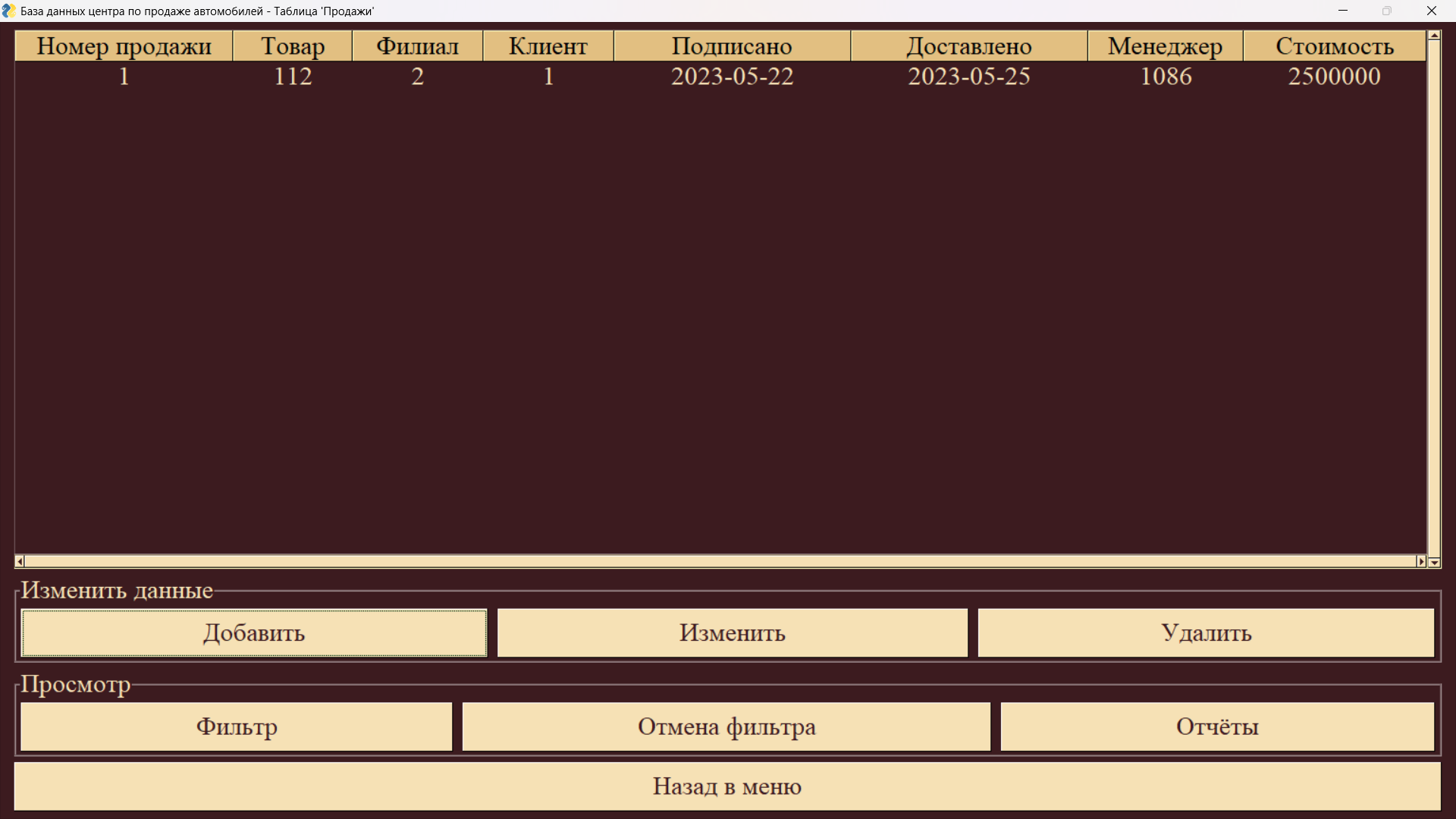


Рисунок 7. Второе окно для таблицы «Продажи»



Рисунок 8. Второе окно для таблицы «Клиенты»



Рисунок 9. Второе окно для таблицы «Поставки»

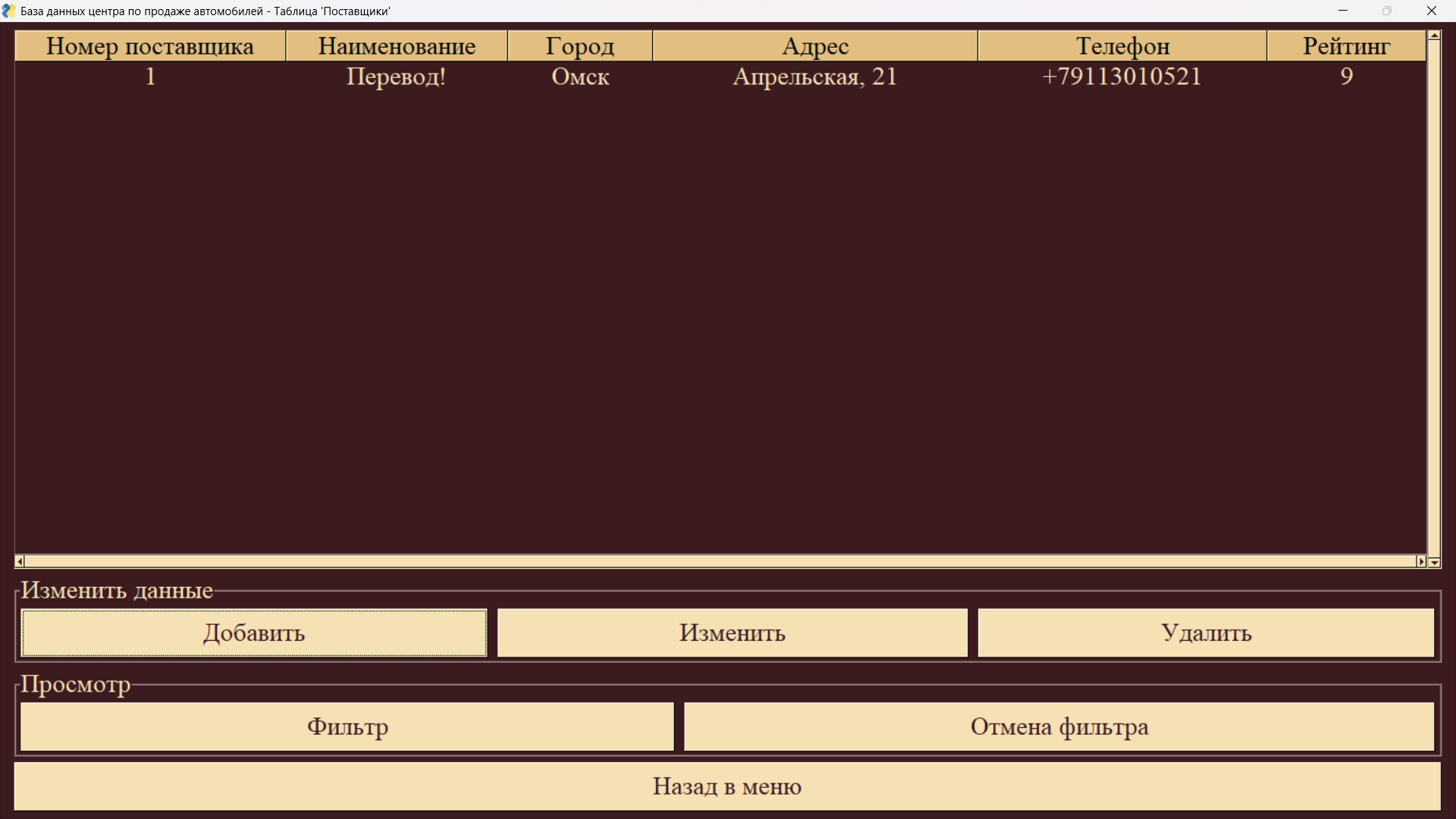


Рисунок 10. Второе окно для таблицы «Поставщики»

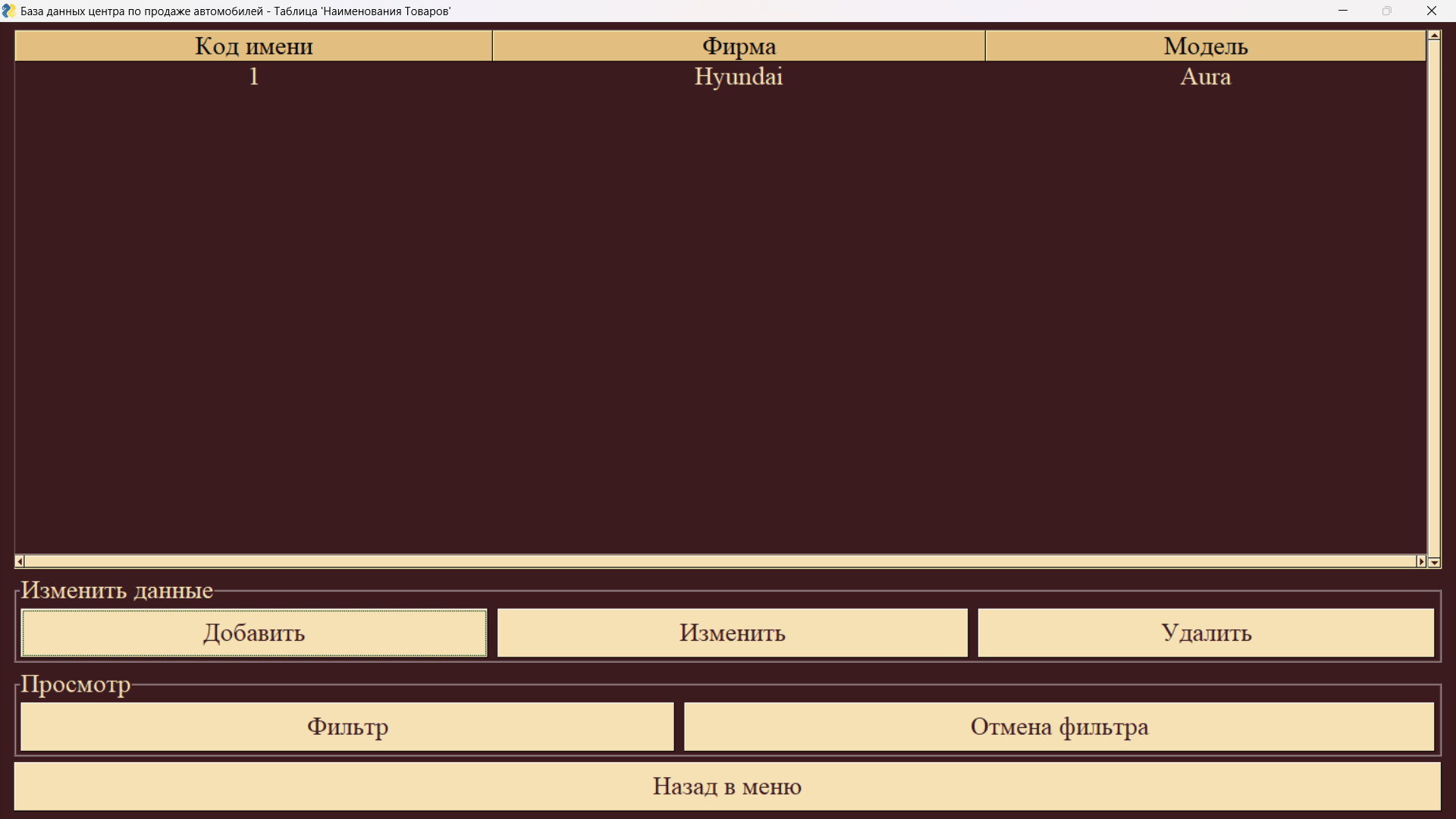


Рисунок 11. Второе окно для таблицы «Наименования Товаров»

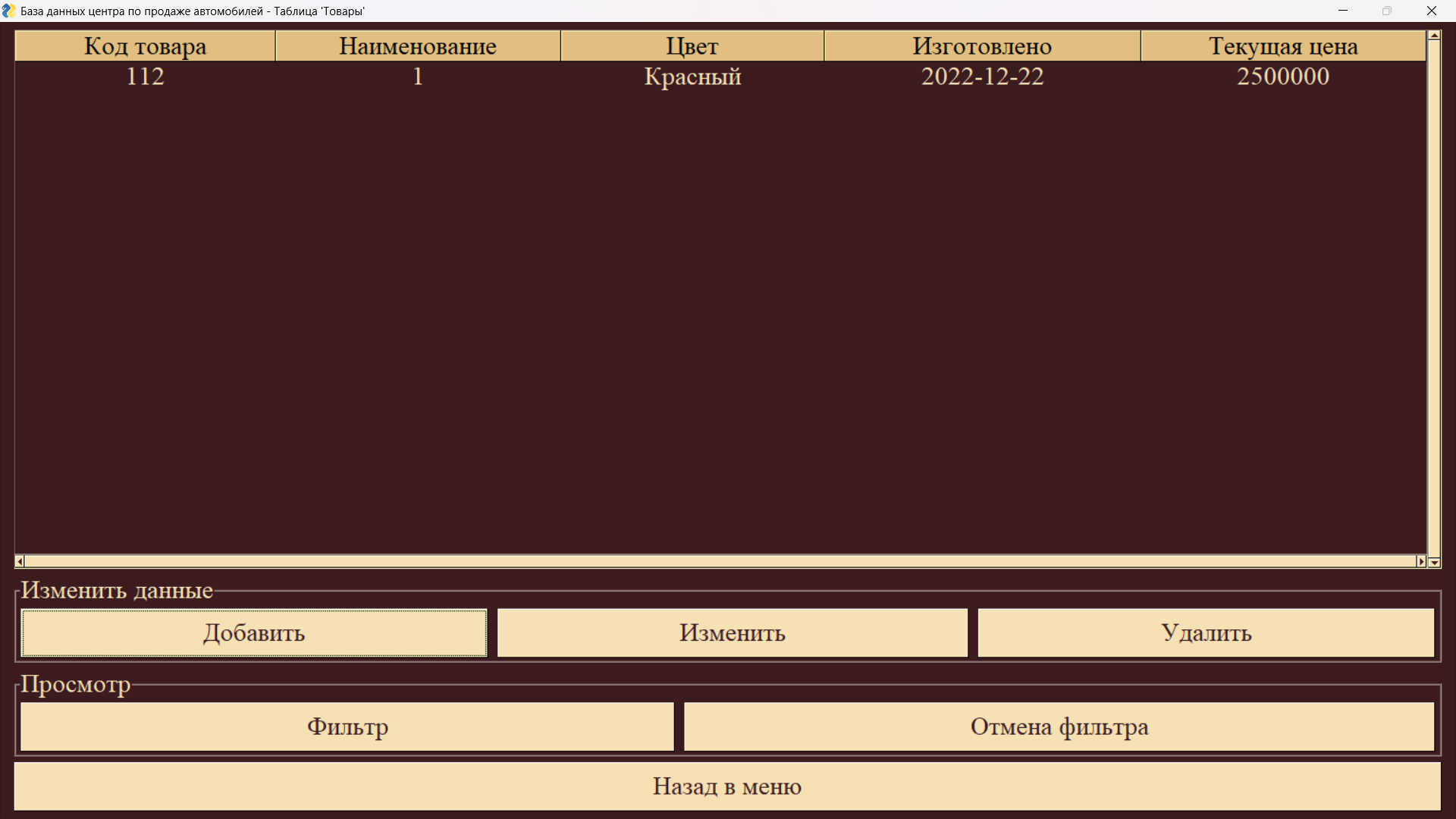


Рисунок 12. Второе окно для таблицы «Товары»



Рисунок 13. Третье окно для добавления данных в таблицу

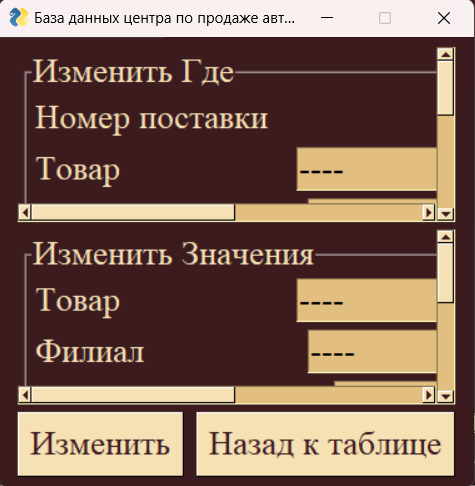


Рисунок 14. Третье окно для изменения данных в таблице

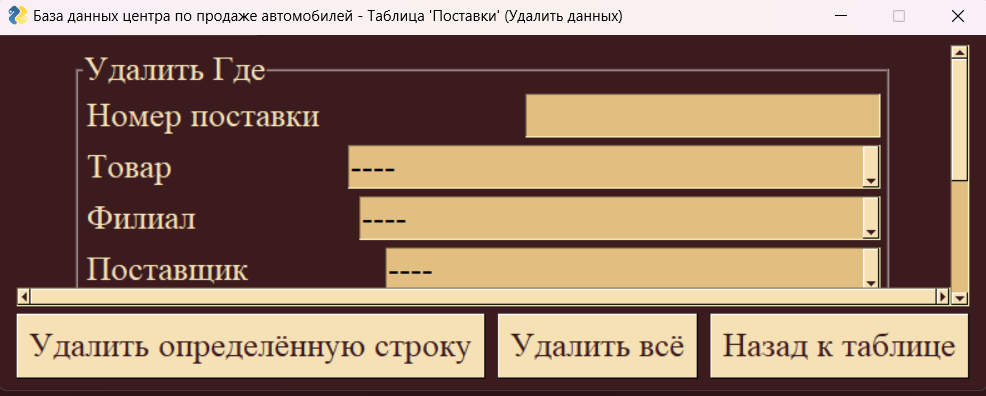


Рисунок 15. Третье окно для удаления данных в таблице



Рисунок 16. Третье окно для фильтрации данных в таблице

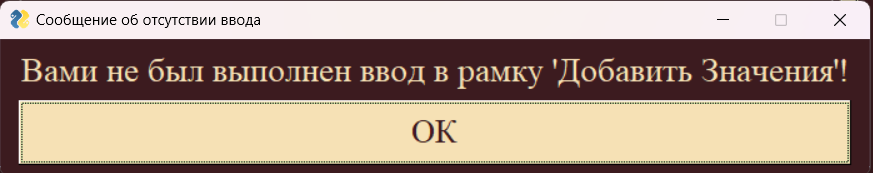


Рисунок 17. Сообщение об отсутствии ввода

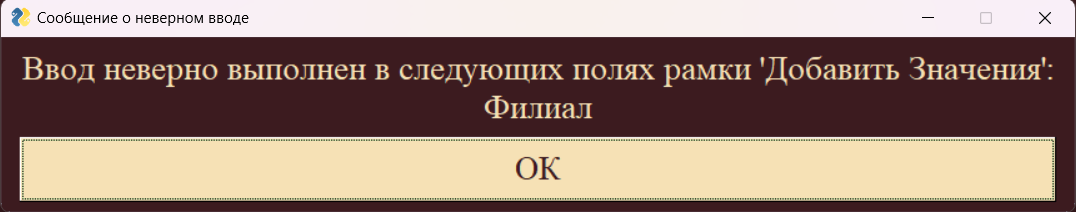


Рисунок 18. Сообщение о неверном вводе

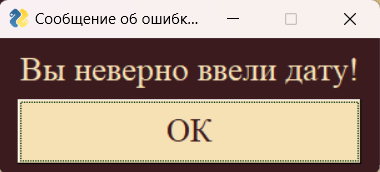


Рисунок 19. Сообщение о нарушении ограничения на ввод дат

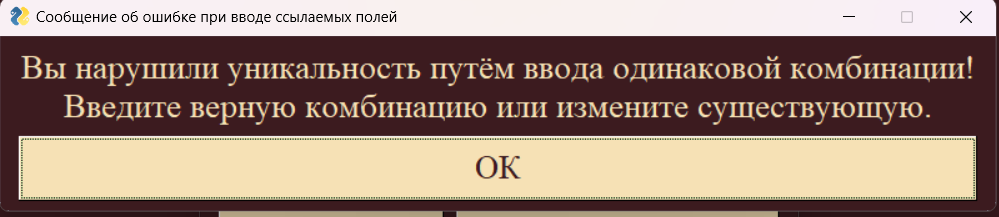


Рисунок 20. Сообщение о нарушении ограничения на ввод комбинации номера филиала и кода товара

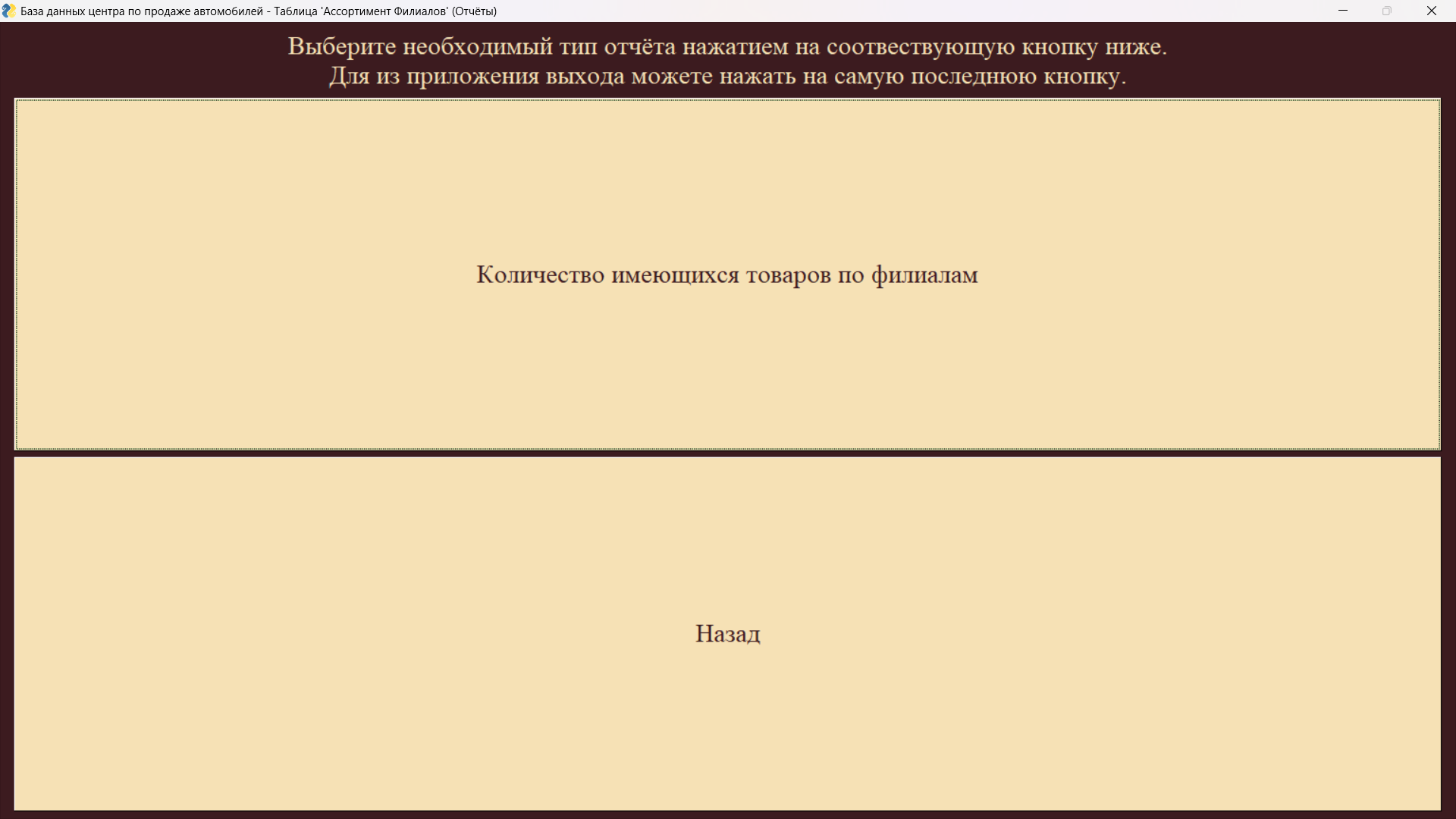


Рисунок 21. Окно набора отчётов для таблицы «Ассортимент Филиалов»

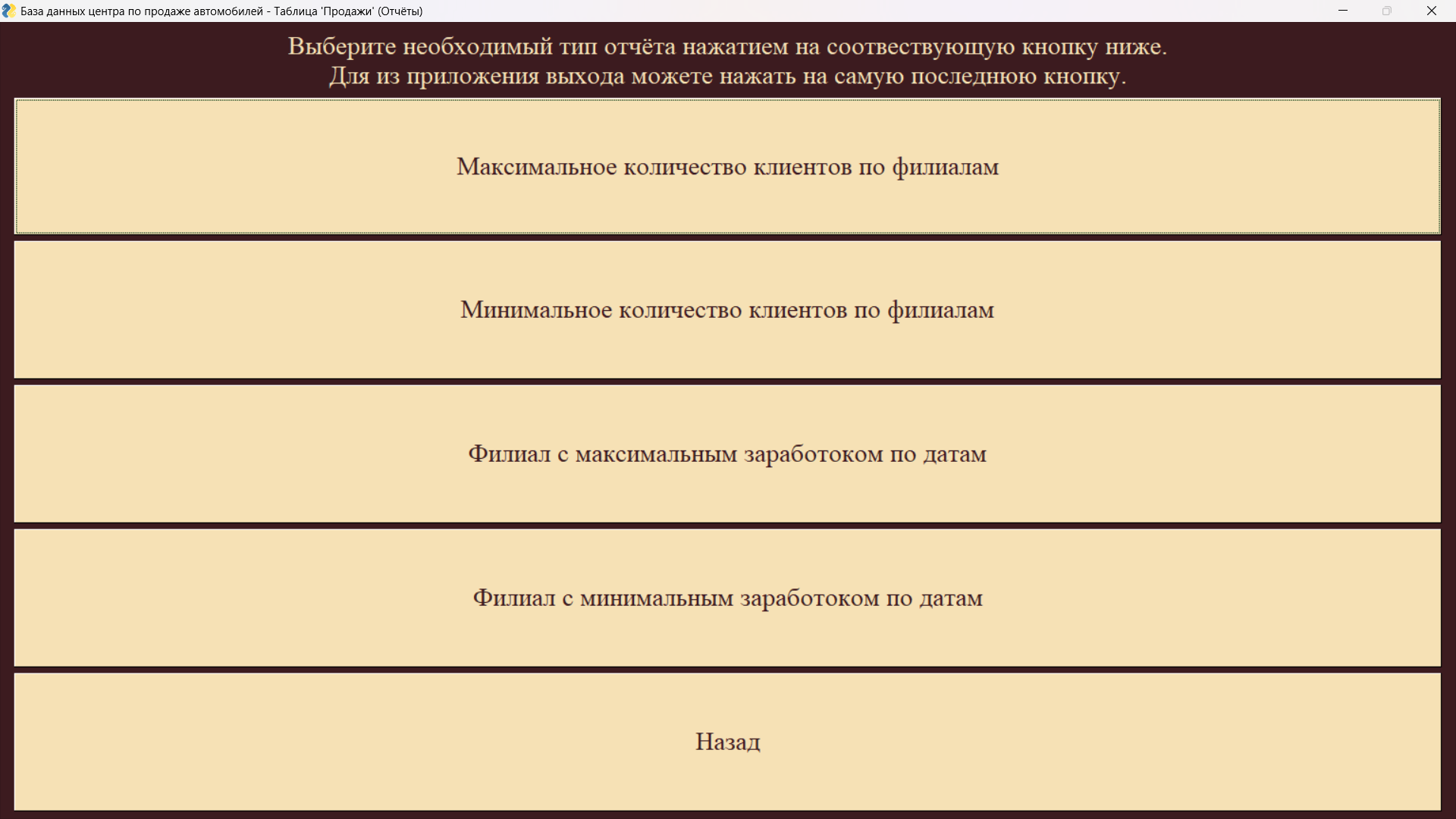


Рисунок 22. Окно набора отчётов для таблицы «Продажи»

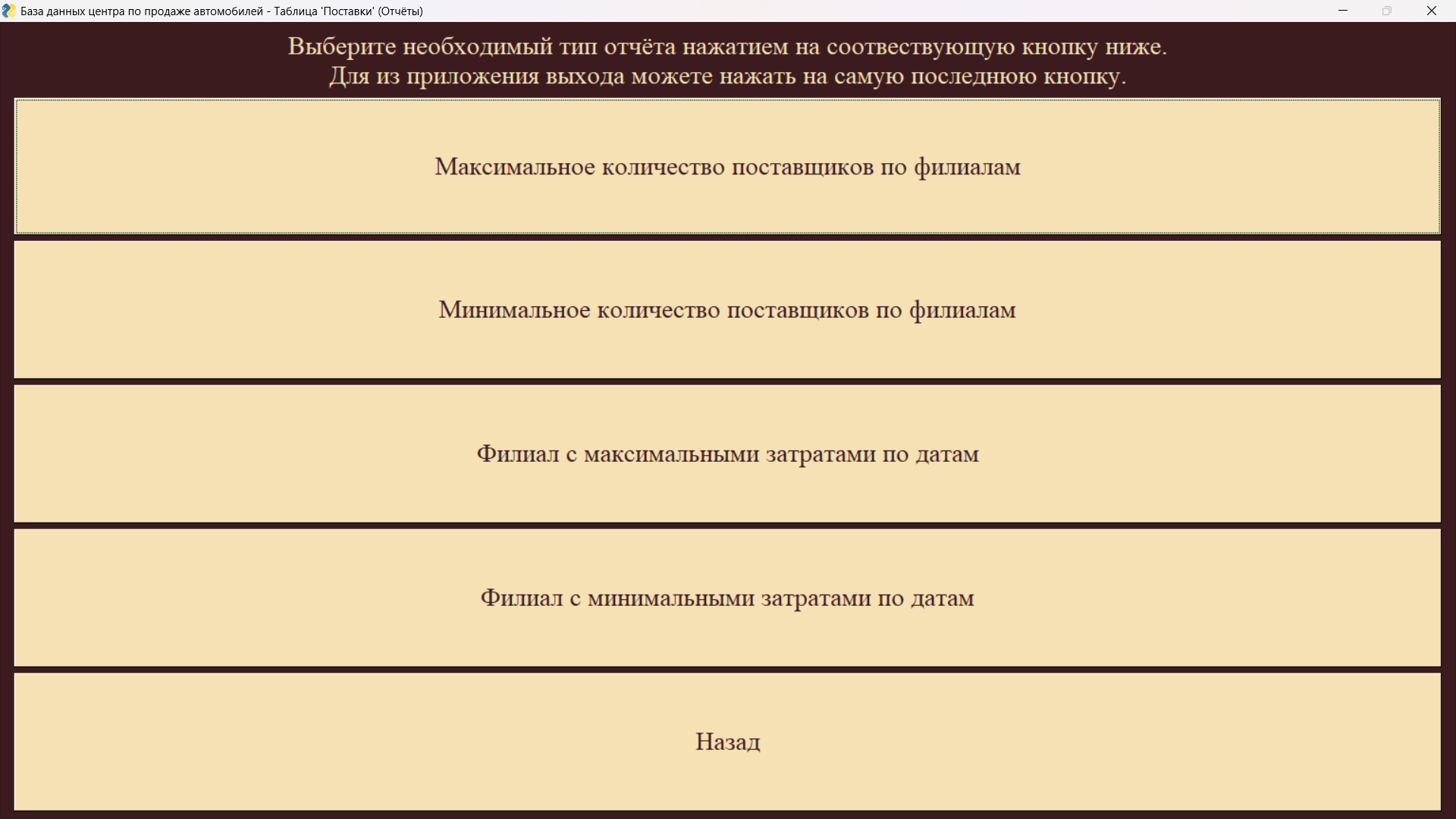


Рисунок 23. Окно набора отчётов для таблицы «Поставки»

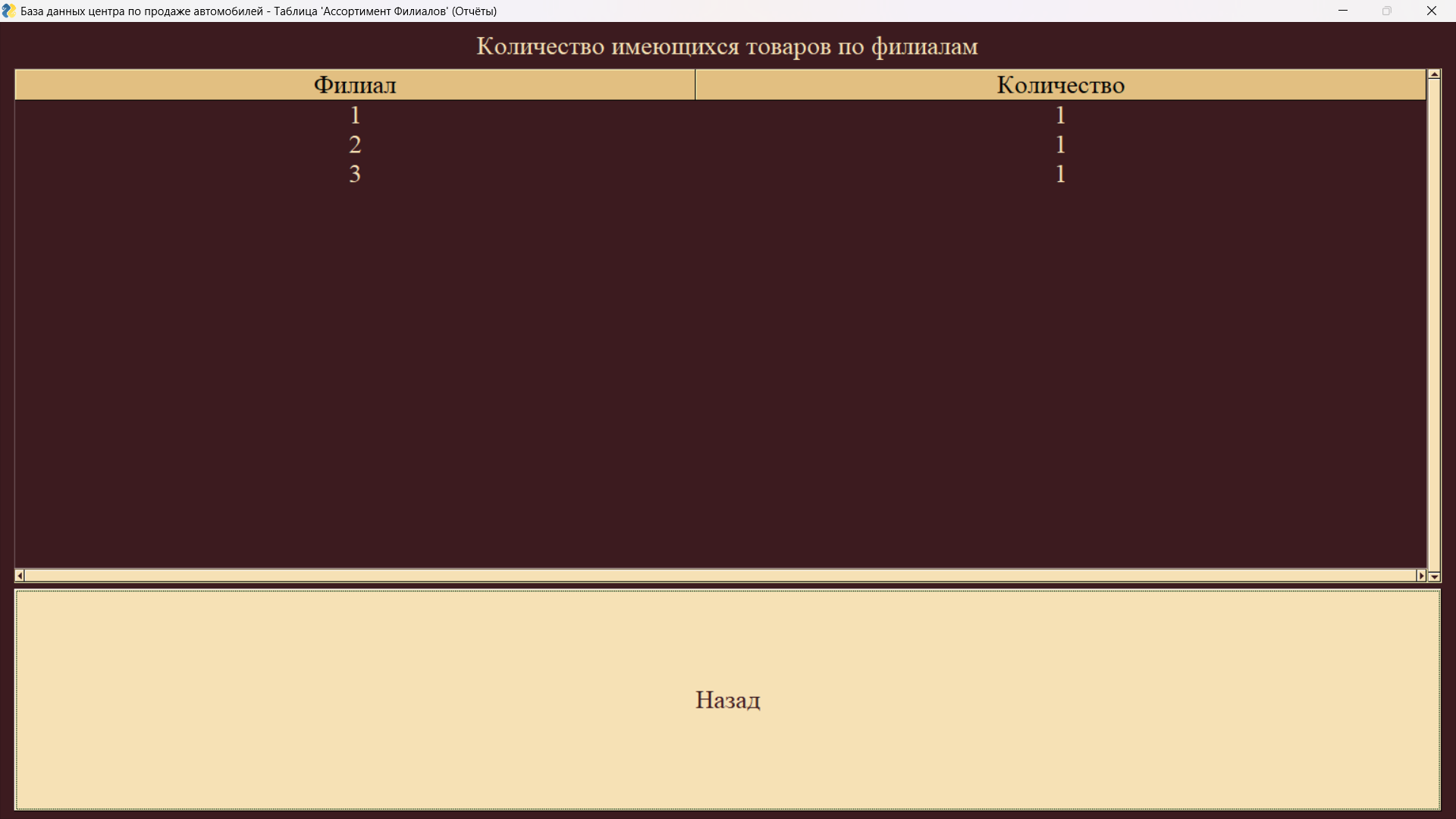


Рисунок 24. Отчёт таблицы «Ассортимент Филиалов», появляющийся при нажатии на первую кнопку

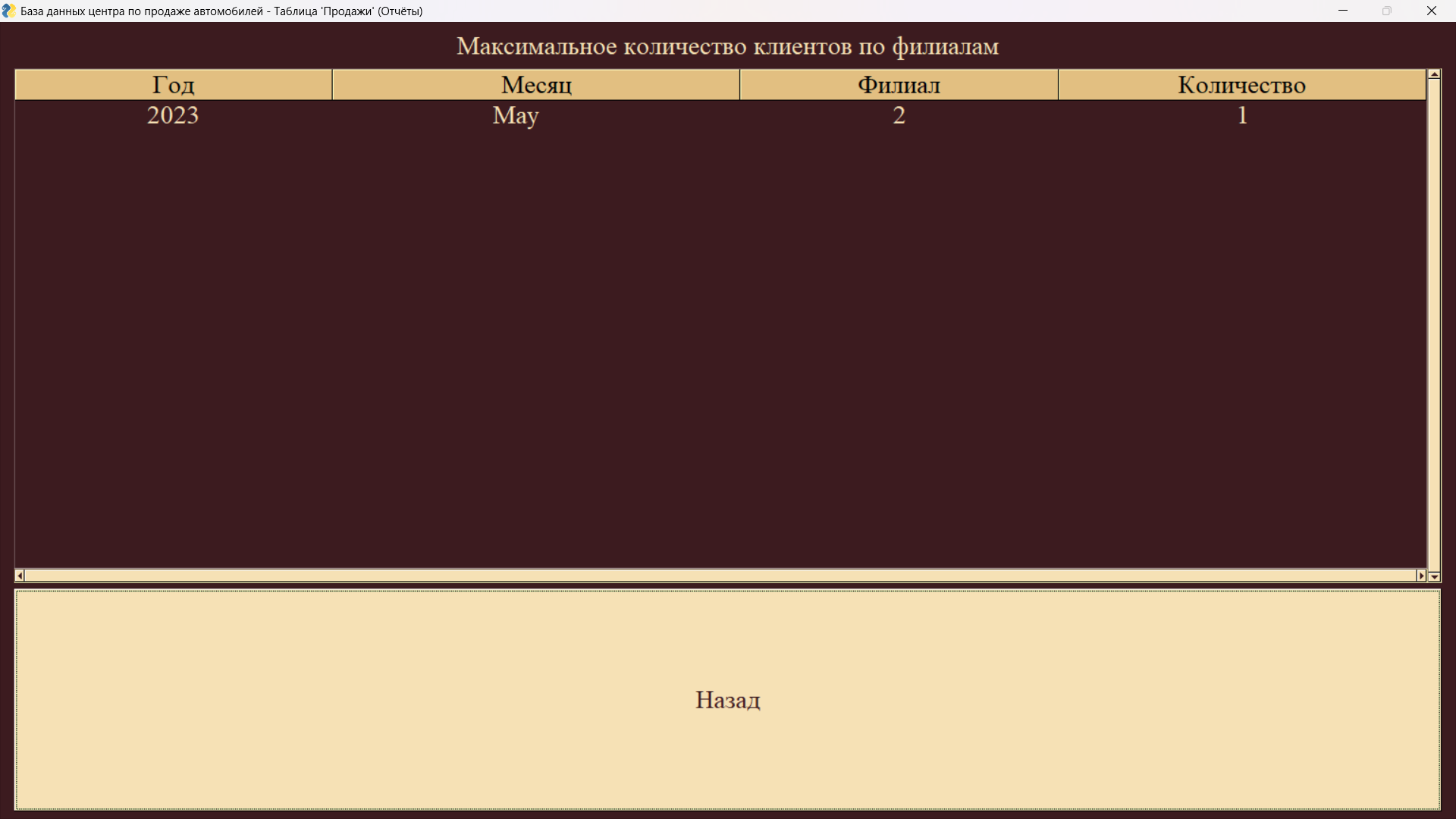


Рисунок 25. Отчёт таблицы «Продажи», появляющийся при нажатии на первую кнопку

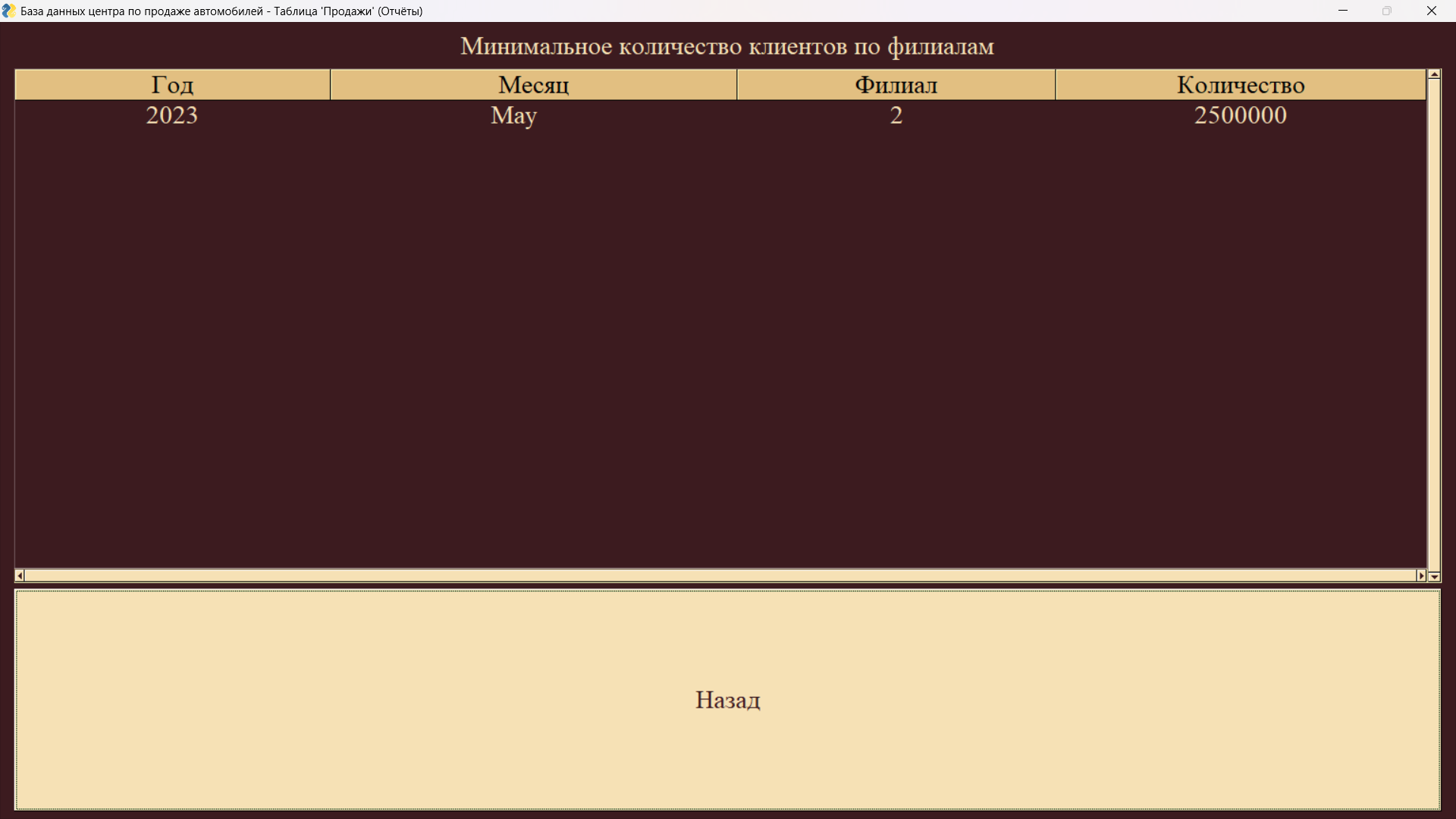


Рисунок 26. Отчёт таблицы «Продажи», появляющийся при нажатии на вторую кнопку

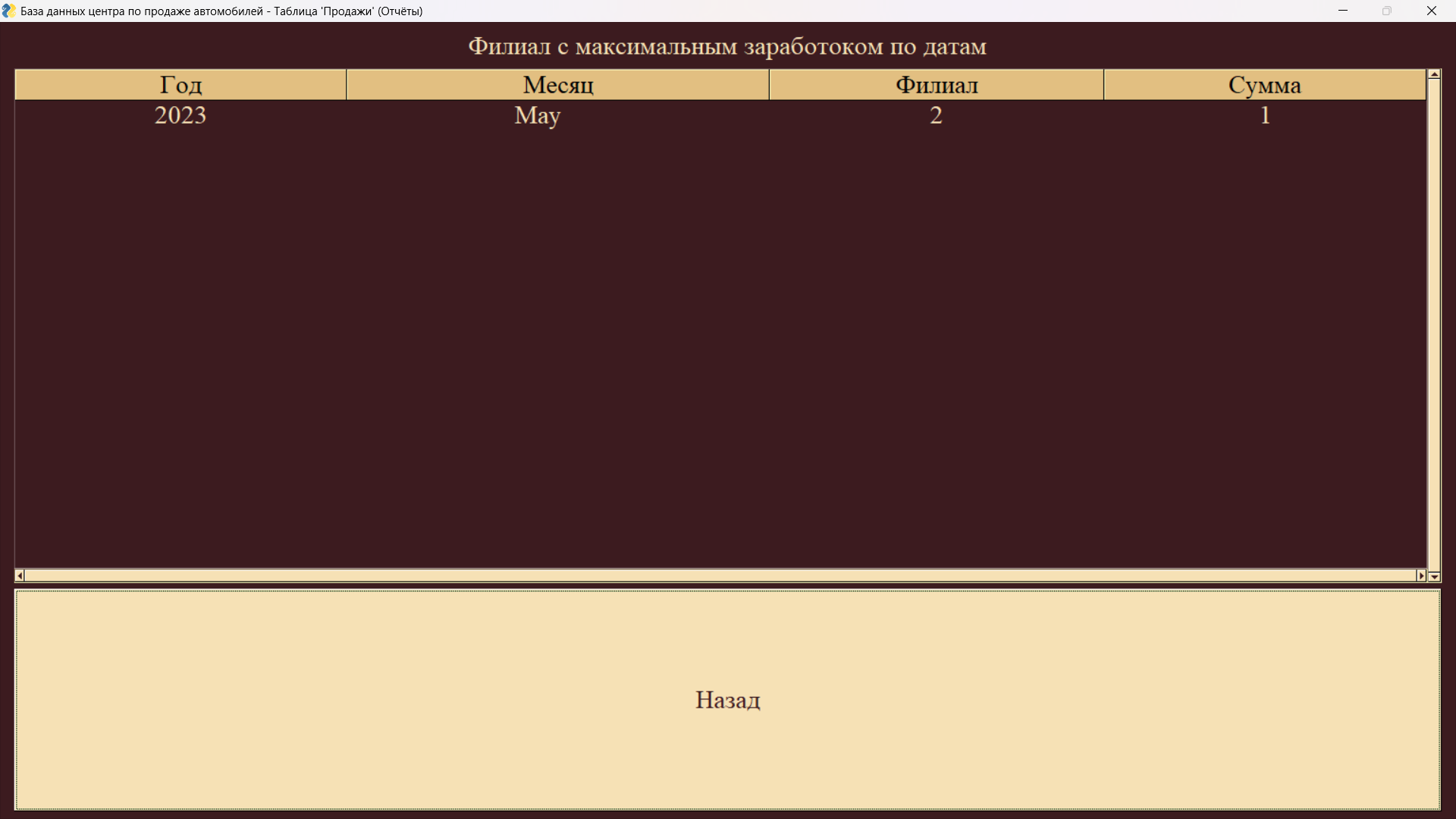
****

Рисунок 27. Отчёт таблицы «Продажи», появляющийся при нажатии на третью кнопку

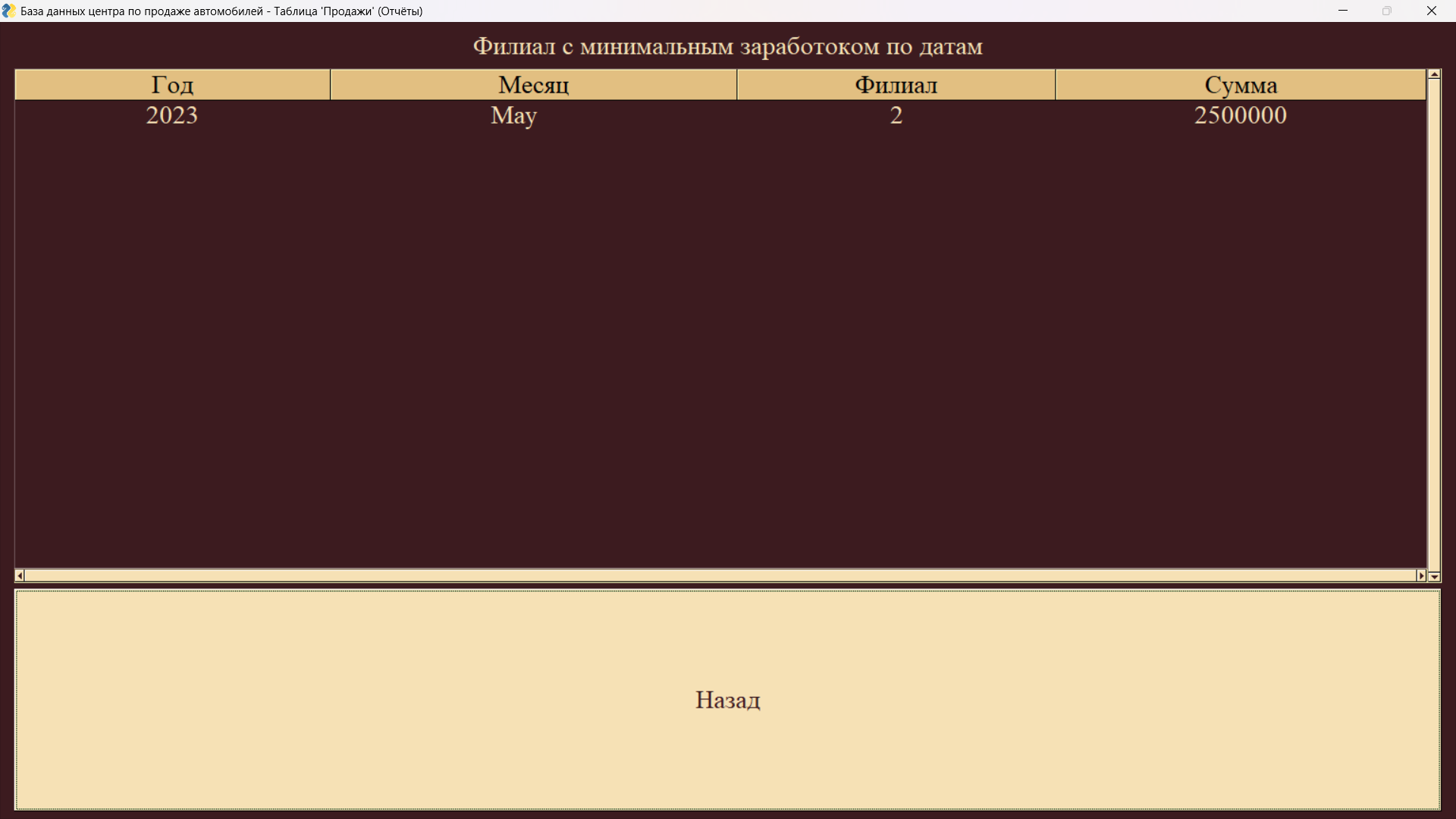
****

Рисунок 28. Отчёт таблицы «Продажи», появляющийся при нажатии на четвёртую кнопку

# ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы было создано приложение, которое позволяет работать с проектированной базой данных, что в итоге даёт возможность эффективным способом работать с данными, связанными с предметной областью.

Данную цель получилось достичь с помощью выполнения следующих задач:

* Описание информационной модели предметной области;
* Разработка ER-диаграммы;
* Программировать объектов базы данных;
* Разработка интерфейса пользователя.

# ССЫЛКИ НА ИСТОЧНИКИ

1. Базы данных в Python: как подключить PostgreSQL и что это такое: онлайн-платформа. – URL: https://ru.hexlet.io/blog/posts/python-postgresql. Дата посещения: (01.05.2023).

2. Сделай то, сделай это, сделай сам: интернет-издание. – URL: https://vc.ru/newtechaudit/413237-sdelay-to-sdelay-eto-sdelay-sam. Дата посещения: (01.05.2023).

3. SQLite, MySQL и PostgreSQL: сравниваем популярные реляционные СУБД: интернет-издание. – URL: https://tproger.ru/translations/sqlite-mysql-postgresql-comparison/. Дата посещения: (19.05.2023).

4. ELEMENT AND FUNCTION CALL REFERENCE: информационный сайт. – URL: https://www.pysimplegui.org/en/latest/call%20reference/#button-element. Дата посещения: (21.05.2023).

### ПРИЛОЖЕНИЕ А

Код приложения представлен ниже:

import PySimpleGUI as Gui

import psycopg2

DBNAME = 'Test'

USER = 'postgres'

PASSWORD = '14032002md'

HOST = 'localhost'

PORT = '5433'

STYLE = "Times New Roman"

ELEMENT\_SIZE = 20

INPUT\_SIZE = 20

DATES\_SIZE = 5

TITLE = "База данных центра по продаже автомобилей"

SCHEMA\_NAME = 'Авто'

DEFAULT\_VALUES = ('', '----')

NULL\_VALUE = None

POST\_NAME = 'Менеджер'

POST\_COLUMN\_NAME = 'Название\_должности'

ERROR\_TEXT = "Ввод неверно выполнен в следующих полях рамки '{}':\n{}"

NO\_INPUT\_TEXT = "Вами не был выполнен ввод в рамку '{}'!"

BOOLEAN\_DEFAULTS = (DEFAULT\_VALUES[1], False, True)

DATES = ('День:', 'Месяц:', 'Год:')

SHOW\_TABLE\_EXECUTE = 'SELECT \* FROM {}.'.format(SCHEMA\_NAME)

INSERT\_TABLE\_EXECUTE = "INSERT INTO {}.{} ({}) VALUES {}"

UPDATE\_TABLE\_EXECUTE = "UPDATE {}.{} SET {}{}"

DELETE\_TABLE\_EXECUTE = "DELETE FROM {}.".format(SCHEMA\_NAME)

ACTION\_LIST = ('Добавить', 'Изменить', 'Удалить')

DELETE\_LIST = ('Удалить определённую строку', 'Удалить всё')

VIEW\_LIST = ('Фильтр', 'Отмена фильтра', 'Отчёты')

FRAME\_NAMES = ('{} Где', '{} Значения')

MONTH\_LIST = []

for index in range(12):

MONTH\_LIST.append(index+1)

DAY\_DICT = {1: 31, 2: [28, 29], 3: 31, 4: 30, 5: 31, 6: 30, 7: 30, 8: 31, 9: 30, 10: 31, 11: 30, 12: 31}

TYPES = ('date', 'boolean', 'integer', 'character varying', 'numeric')

REPORTS\_TABLES = ['Ассортимент\_Филиалов', 'Продажи', 'Поставки']

REPORTS\_TYPES = {'Ассортимент\_Филиалов': ['Количество имеющихся товаров по филиалам'],

'Продажи': ['Максимальное количество клиентов по филиалам', 'Минимальное количество клиентов по филиалам',

'Филиал с максимальным заработоком по датам', 'Филиал с минимальным заработоком по датам'],

'Поставки': ['Максимальное количество поставщиков по филиалам',

'Минимальное количество поставщиков по филиалам',

'Филиал с максимальными затратами по датам', 'Филиал с минимальными затратами по датам']}

with psycopg2.connect(dbname=DBNAME, user=USER, password=PASSWORD, host=HOST, port=PORT) as connection:

with connection.cursor() as cursor:

TABLES\_LIST = []

DATE\_COLUMNS = []

BOOLEAN\_COLUMNS = []

INTEGER\_COLUMNS = []

VARCHAR\_COLUMNS = []

NUMERIC\_COLUMNS = []

PRIMARY\_KEY\_COLUMNS = []

FOREIGN\_KEY\_COLUMNS = []

SPECIAL\_POST\_COLUMNS = []

NO\_INSERT\_COLUMNS = ['Продажи.Стоимость']

VARCHAR\_LENGTH = {}

FOREIGN\_KEY\_TABLE = {}

FOREIGN\_KEY\_TABLE\_COLUMN = {}

cursor.execute("SELECT tablename\n"

"FROM pg\_catalog.pg\_tables\n"

"WHERE schemaname=%s", (SCHEMA\_NAME,))

tables = cursor.fetchall()

for elem in tables:

TABLES\_LIST.append(elem[0])

cursor.execute("SELECT table\_name, column\_name\n"

"FROM INFORMATION\_SCHEMA.key\_column\_usage\n"

"WHERE table\_schema=%s AND constraint\_name IN (\n"

" SELECT constraint\_name\n"

" FROM INFORMATION\_SCHEMA.table\_constraints\n"

" WHERE constraint\_type=%s)", (SCHEMA\_NAME, 'PRIMARY KEY'))

primary\_keys = cursor.fetchall()

for index in range(len(primary\_keys)):

primary\_key = '{}.{}'.format(primary\_keys[index][0], primary\_keys[index][1])

PRIMARY\_KEY\_COLUMNS.append(primary\_key)

cursor.execute("WITH con\_1 AS (\n"

" SELECT constraint\_name, table\_name, column\_name\n"

" FROM INFORMATION\_SCHEMA.key\_column\_usage\n"

" WHERE table\_schema=%s AND constraint\_name IN (\n"

" SELECT constraint\_name\n"

" FROM INFORMATION\_SCHEMA.table\_constraints\n"

" WHERE constraint\_type=%s)), con\_2 AS(\n"

" SELECT constraint\_name, table\_name, column\_name\n"

" FROM INFORMATION\_SCHEMA.constraint\_column\_usage\n"

" WHERE constraint\_schema=%s)\n"

"SELECT con\_1.table\_name, con\_1.column\_name, con\_2.table\_name, con\_2.column\_name\n"

"FROM con\_1 INNER JOIN con\_2 USING(constraint\_name)", (SCHEMA\_NAME, 'FOREIGN KEY', SCHEMA\_NAME,))

col\_keys = cursor.fetchall()

for index in range(len(col\_keys)):

foreign\_key = '{}.{}'.format(col\_keys[index][0], col\_keys[index][1])

primary\_key = '{}.{}'.format(col\_keys[index][2], col\_keys[index][3])

FOREIGN\_KEY\_COLUMNS.append(foreign\_key)

if col\_keys[index][1] == POST\_NAME:

SPECIAL\_POST\_COLUMNS.append(foreign\_key)

FOREIGN\_KEY\_TABLE[foreign\_key] = col\_keys[index][2]

FOREIGN\_KEY\_TABLE\_COLUMN[foreign\_key] = primary\_key

for one\_type in TYPES:

cursor.execute("SELECT table\_name, column\_name\n"

"FROM INFORMATION\_SCHEMA.COLUMNS\n"

"WHERE table\_schema=%s AND data\_type=%s", (SCHEMA\_NAME, one\_type))

data = cursor.fetchall()

if one\_type == TYPES[0]:

for elem in data:

DATE\_COLUMNS.append('{}.{}'.format(elem[0], elem[1]))

elif one\_type == TYPES[1]:

for elem in data:

BOOLEAN\_COLUMNS.append('{}.{}'.format(elem[0], elem[1]))

elif one\_type == TYPES[2]:

for elem in data:

INTEGER\_COLUMNS.append('{}.{}'.format(elem[0], elem[1]))

elif one\_type == TYPES[3]:

for elem in data:

VARCHAR\_COLUMNS.append('{}.{}'.format(elem[0], elem[1]))

cursor.execute("SELECT table\_name, column\_name, character\_maximum\_length\n"

"FROM INFORMATION\_SCHEMA.COLUMNS\n"

"WHERE table\_schema = '{}' AND data\_type = '{}' "

"AND character\_maximum\_length <> {}".format(SCHEMA\_NAME, one\_type, 0))

varchar\_length = cursor.fetchall()

for index in range(len(varchar\_length)):

name = '{}.{}'.format(varchar\_length[index][0], varchar\_length[index][1])

value = int(varchar\_length[index][2])

VARCHAR\_LENGTH[name] = value

elif one\_type == TYPES[4]:

for elem in data:

NUMERIC\_COLUMNS.append('{}.{}'.format(elem[0], elem[1]))

def db\_get(operation):

with psycopg2.connect(dbname=DBNAME, user=USER, password=PASSWORD, host=HOST, port=PORT) as connection:

with connection.cursor() as cursor:

cursor.execute(operation)

get\_column\_names = [desc[0] for desc in cursor.description]

get\_values = cursor.fetchall()

return get\_values

def db\_edit(operation):

try:

with psycopg2.connect(dbname=DBNAME, user=USER, password=PASSWORD, host=HOST, port=PORT) as connection:

with connection.cursor() as cursor:

cursor.execute(operation)

except psycopg2.errors.CheckViolation:

check\_error()

except psycopg2.errors.UniqueViolation:

unique\_error()

def no\_input\_error(frame\_name):

text = NO\_INPUT\_TEXT.format(frame\_name)

no\_error\_title = 'Сообщение об отсутствии ввода'

no\_error\_layout = []

no\_error\_text = Gui.Text(text=text, justification='center', key=0)

no\_error\_button = Gui.Button(button\_text='ОК', key=1, expand\_x=True)

no\_error\_layout = [[no\_error\_text], [no\_error\_button]]

no\_error\_window = Gui.Window(title=no\_error\_title, layout=no\_error\_layout, modal=True).Finalize()

while True:

event, values = no\_error\_window.read()

if event == 1:

break

no\_error\_window.close()

def check\_error():

text = 'Вы неверно ввели дату!'

check\_error\_title = 'Сообщение об ошибке при вводе даты'

no\_error\_text = Gui.Text(text=text, justification='center', key=0)

no\_error\_button = Gui.Button(button\_text='ОК', key=1, expand\_x=True)

check\_error\_layout = [[no\_error\_text], [no\_error\_button]]

check\_error\_window = Gui.Window(title=check\_error\_title, layout=check\_error\_layout, modal=True).Finalize()

while True:

event, values = check\_error\_window.read()

if event == 1:

break

check\_error\_window.close()

def unique\_error():

text = "Вы нарушили уникальность путём ввода одинаковой комбинации!\n" \

"Введите верную комбинацию или измените существующую."

check\_error\_title = 'Сообщение об ошибке при вводе ссылаемых полей'

no\_error\_text = Gui.Text(text=text, justification='center', key=0)

no\_error\_button = Gui.Button(button\_text='ОК', key=1, expand\_x=True)

check\_error\_layout = [[no\_error\_text], [no\_error\_button]]

check\_error\_window = Gui.Window(title=check\_error\_title, layout=check\_error\_layout, modal=True).Finalize()

while True:

event, values = check\_error\_window.read()

if event == 1:

break

check\_error\_window.close()

def input\_error(frame\_name\_1, problems\_1, frame\_name\_2, problems\_2):

text = ''

edit\_problems\_1 = []

edit\_problems\_2 = []

for index in range(len(problems\_1)):

edit\_problems\_1.append(problems\_1[index].replace('\_', ' '))

for index in range(len(problems\_2)):

edit\_problems\_2.append(problems\_2[index].replace('\_', ' '))

frame\_problems\_1 = ', '.join(edit\_problems\_1)

frame\_problems\_2 = ', '.join(edit\_problems\_2)

input\_error\_title = 'Сообщение о неверном вводе'

if (len(problems\_1) != 0) and (len(problems\_2) != 0):

add\_text = '{}\n{}'.format(ERROR\_TEXT, ERROR\_TEXT)

text = add\_text.format(frame\_name\_1, frame\_problems\_1, frame\_name\_2, frame\_problems\_2)

elif len(problems\_1) != 0:

text = ERROR\_TEXT.format(frame\_name\_1, frame\_problems\_1)

elif len(problems\_2) != 0:

text = ERROR\_TEXT.format(frame\_name\_2, frame\_problems\_2)

input\_error\_text = Gui.Text(text=text, justification='center', key=0)

input\_error\_button = Gui.Button(button\_text='ОК', key=1, expand\_x=True)

input\_error\_layout = [[input\_error\_text], [input\_error\_button]]

input\_error\_window = Gui.Window(title=input\_error\_title, layout=input\_error\_layout, modal=True).Finalize()

while True:

event, values = input\_error\_window.read()

if event == 1:

break

input\_error\_window.close()

def values\_frame(elements\_count, table\_name, column\_names):

date\_count = 0

insert\_count = 0

primary\_key\_count = 0

frame\_layout = []

insert\_column\_names = []

for index in range(len(column\_names)):

index\_number = 2\*date\_count+elements\_count+index-insert\_count-primary\_key\_count

full\_name = '{}.{}'.format(table\_name, column\_names[index])

if full\_name in PRIMARY\_KEY\_COLUMNS:

if full\_name not in INTEGER\_COLUMNS:

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

new\_input = Gui.Input(default\_text=DEFAULT\_VALUES[0],

key=index\_number)

frame\_layout.append([new\_text, new\_input])

else:

primary\_key\_count += 1

elif full\_name in FOREIGN\_KEY\_COLUMNS:

keys = [(DEFAULT\_VALUES[1])]

if full\_name not in SPECIAL\_POST\_COLUMNS:

get\_values = 'SELECT {}\n' \

'FROM {}'.format(FOREIGN\_KEY\_TABLE\_COLUMN[full\_name],

'{}.{}'.format(SCHEMA\_NAME, FOREIGN\_KEY\_TABLE[full\_name]))

else:

foreign\_key = ''

primary\_key = ''

primary\_key\_list = []

for elem in FOREIGN\_KEY\_COLUMNS:

if FOREIGN\_KEY\_TABLE[full\_name] in elem:

foreign\_key = elem

primary\_key = FOREIGN\_KEY\_TABLE\_COLUMN[elem]

primary\_key\_list = primary\_key.split('.')

get\_values = 'SELECT {}\n' \

'FROM {} INNER JOIN {} ON {}={}\n' \

'WHERE {}=%s'.format(FOREIGN\_KEY\_TABLE\_COLUMN[full\_name],

'{}.{}'.format(SCHEMA\_NAME, FOREIGN\_KEY\_TABLE[full\_name]),

'{}.{}'.format(SCHEMA\_NAME, primary\_key\_list[0]), foreign\_key,

primary\_key, POST\_COLUMN\_NAME)

values = db\_get(get\_values.replace('%s', ("'{}'".format(POST\_NAME))))

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

for elem in values:

keys.append(elem)

new\_combo = Gui.Combo(keys, default\_value=DEFAULT\_VALUES[1], size=(INPUT\_SIZE, 1), expand\_x=True,

readonly=False, key=index\_number)

frame\_layout.append([new\_text, new\_combo])

elif full\_name in DATE\_COLUMNS:

date\_layout = []

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

for times in range(len(DATES)):

date\_text = Gui.Text(text=DATES[times])

date\_layout.append(date\_text)

date\_input = Gui.Input(default\_text=DEFAULT\_VALUES[0], size=(DATES\_SIZE, 1), key=index\_number+times)

date\_layout.append(date\_input)

date\_count += 1

date\_frame = Gui.Frame("", [date\_layout])

frame\_layout.append([new\_text, date\_frame])

elif full\_name in BOOLEAN\_COLUMNS:

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

new\_combo = Gui.Combo(BOOLEAN\_DEFAULTS, size=(INPUT\_SIZE, 1), default\_value=BOOLEAN\_DEFAULTS[0],

expand\_x=True, readonly=False, key=index\_number)

frame\_layout.append([new\_text, new\_combo])

elif full\_name in NUMERIC\_COLUMNS:

if full\_name not in NO\_INSERT\_COLUMNS:

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

new\_input = Gui.Input(default\_text='', size=(INPUT\_SIZE, 1), key=index\_number)

frame\_layout.append([new\_text, new\_input])

else:

insert\_count += 1

elif (full\_name in VARCHAR\_COLUMNS) or (full\_name in INTEGER\_COLUMNS):

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

new\_input = Gui.Input(default\_text=DEFAULT\_VALUES[0], size=(INPUT\_SIZE, 1), key=index\_number)

frame\_layout.append([new\_text, new\_input])

if (full\_name not in PRIMARY\_KEY\_COLUMNS) or (full\_name not in INTEGER\_COLUMNS):

if full\_name not in NO\_INSERT\_COLUMNS:

insert\_column\_names.append(column\_names[index])

return frame\_layout, insert\_column\_names, 2\*date\_count+elements\_count+(

len(column\_names)-primary\_key\_count-insert\_count)

def where\_frame(table\_name, column\_names):

date\_count = 0

frame\_layout = []

for index in range(len(column\_names)):

index\_number = 2\*date\_count+index

full\_name = '{}.{}'.format(table\_name, column\_names[index])

if full\_name in FOREIGN\_KEY\_COLUMNS:

keys = [(DEFAULT\_VALUES[1]), NULL\_VALUE]

if full\_name not in SPECIAL\_POST\_COLUMNS:

get\_values = 'SELECT {}\n' \

'FROM {}'.format(FOREIGN\_KEY\_TABLE\_COLUMN[full\_name],

'{}.{}'.format(SCHEMA\_NAME, FOREIGN\_KEY\_TABLE[full\_name]))

else:

foreign\_key = ''

primary\_key = ''

primary\_key\_list = []

for elem in FOREIGN\_KEY\_COLUMNS:

if FOREIGN\_KEY\_TABLE[full\_name] in elem:

foreign\_key = elem

primary\_key = FOREIGN\_KEY\_TABLE\_COLUMN[elem]

primary\_key\_list = primary\_key.split('.')

get\_values = 'SELECT {}\n' \

'FROM {} INNER JOIN {} ON {}={}\n' \

'WHERE {}=%s'.format(FOREIGN\_KEY\_TABLE\_COLUMN[full\_name],

'{}.{}'.format(SCHEMA\_NAME, FOREIGN\_KEY\_TABLE[full\_name]),

'{}.{}'.format(SCHEMA\_NAME, primary\_key\_list[0]), foreign\_key,

primary\_key, POST\_COLUMN\_NAME)

values = db\_get(get\_values.replace('%s', ("'{}'".format(POST\_NAME))))

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

for elem in values:

keys.append(elem)

new\_combo = Gui.Combo(keys, default\_value=DEFAULT\_VALUES[1], size=(INPUT\_SIZE, 1), expand\_x=True,

readonly=False, key=index\_number)

frame\_layout.append([new\_text, new\_combo])

elif full\_name in DATE\_COLUMNS:

date\_layout = []

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

for times in range(len(DATES)):

date\_text = Gui.Text(text=DATES[times])

date\_layout.append(date\_text)

date\_input = Gui.Input(default\_text=DEFAULT\_VALUES[0], size=(DATES\_SIZE, 1), key=index\_number+times)

date\_layout.append(date\_input)

date\_count += 1

date\_frame = Gui.Frame("", [date\_layout])

frame\_layout.append([new\_text, date\_frame])

elif full\_name in BOOLEAN\_COLUMNS:

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

new\_combo = Gui.Combo(BOOLEAN\_DEFAULTS, size=(INPUT\_SIZE, 1), default\_value=BOOLEAN\_DEFAULTS[0],

expand\_x=True, readonly=False, key=index\_number)

frame\_layout.append([new\_text, new\_combo])

elif full\_name in NUMERIC\_COLUMNS:

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

new\_input = Gui.Input(default\_text=DEFAULT\_VALUES[0], size=(INPUT\_SIZE, 1), key=index\_number)

frame\_layout.append([new\_text, new\_input])

elif (full\_name in INTEGER\_COLUMNS) or (full\_name in VARCHAR\_COLUMNS):

new\_text = Gui.Text(text=column\_names[index].replace("\_", " "), expand\_x=True)

new\_input = Gui.Input(default\_text=DEFAULT\_VALUES[0], size=(INPUT\_SIZE, 1), key=index\_number)

frame\_layout.append([new\_text, new\_input])

return frame\_layout, 2\*date\_count+len(column\_names)

def values\_frame\_input\_check(elements\_count, table\_name, values, column\_names):

date\_count = 0

insert\_count = 0

primary\_key\_count = 0

check\_date = True

dates = []

problems = []

numbers\_check = []

integer\_check = []

numeric\_check = []

insert\_values = []

for i in range(10):

numbers\_check.append(str(i))

integer\_check.append(str(i))

numeric\_check.append(str(i))

integer\_check.append(' ')

numeric\_check.append('.')

for index in range(len(column\_names)):

index\_number = 2\*date\_count+elements\_count+index-insert\_count-primary\_key\_count

full\_name = table\_name + '.' + column\_names[index]

if full\_name in PRIMARY\_KEY\_COLUMNS:

if full\_name not in INTEGER\_COLUMNS:

repeat\_count = db\_get("SELECT COUNT({})\n"

"FROM {}\n"

"WHERE {}='{}'".format(column\_names[index],

'{}.{}'.format(SCHEMA\_NAME, table\_name),

column\_names[index], values[index\_number]))[0][0]

if int(repeat\_count) != 0:

problems.append(column\_names[index])

else:

primary\_key\_count += 1

elif full\_name in FOREIGN\_KEY\_COLUMNS:

foreign\_keys = db\_get('SELECT {}\n'

'FROM {}'.format(FOREIGN\_KEY\_TABLE\_COLUMN[full\_name],

'{}.{}'.format(SCHEMA\_NAME, FOREIGN\_KEY\_TABLE[full\_name])))

if (values[index\_number] not in foreign\_keys) and (values[index\_number] not in DEFAULT\_VALUES):

problems.append(column\_names[index])

elif full\_name in BOOLEAN\_COLUMNS:

if values[index\_number] not in BOOLEAN\_DEFAULTS:

problems.append(column\_names[index])

elif full\_name in VARCHAR\_LENGTH:

if len(values[index\_number]) > VARCHAR\_LENGTH[full\_name]:

problems.append(column\_names[index])

elif full\_name in NUMERIC\_COLUMNS:

if full\_name not in NO\_INSERT\_COLUMNS:

error\_count = sum(values[index\_number][element] not in numeric\_check

for element in range(len(values[index\_number])))

if error\_count != 0:

problems.append(column\_names[index])

else:

if values[index\_number].count('.') > 1:

problems.append(column\_names[index])

else:

insert\_count += 1

elif full\_name in INTEGER\_COLUMNS:

error\_count = sum(values[index\_number][element] not in integer\_check

for element in range(len(values[index\_number])))

if error\_count != 0:

problems.append(column\_names[index])

elif full\_name in DATE\_COLUMNS:

check\_date\_count = sum(values[index\_number+times] == DEFAULT\_VALUES[0]

for times in range(len(DATES)))

if (check\_date\_count > 0) and (check\_date\_count < 3):

problems.append(column\_names[index])

elif check\_date\_count == 0:

date = []

for times in range(len(DATES)):

error\_count = sum(values[index\_number+times][element] not in numbers\_check

for element in range(len(values[index\_number+times])))

if error\_count != 0:

problems.append(column\_names[index])

check\_date = False

break

else:

date.append(int(values[index\_number+times]))

if check\_date:

if date[1] not in MONTH\_LIST:

problems.append(column\_names[index])

else:

if type(DAY\_DICT[date[1]]) is list:

if date[2] % 4 == 0:

max\_day = DAY\_DICT[date[1]][1]

else:

max\_day = DAY\_DICT[date[1]][0]

if (date[0] > max\_day) and (date[0] == 0):

problems.append(column\_names[index])

else:

if (date[0] > DAY\_DICT[date[1]]) or (date[0] == 0):

problems.append(column\_names[index])

if column\_names[index] not in problems:

dates.append([date][0])

date\_count += 1

if full\_name in FOREIGN\_KEY\_COLUMNS:

if full\_name in INTEGER\_COLUMNS:

insert\_values.append(values[index\_number][0])

elif full\_name in VARCHAR\_COLUMNS:

insert\_values.append('{}'.format(values[index\_number][0]))

elif (full\_name in PRIMARY\_KEY\_COLUMNS) and (full\_name in VARCHAR\_COLUMNS):

insert\_values.append(values[index\_number])

elif full\_name in DATE\_COLUMNS:

if values[index\_number] not in DEFAULT\_VALUES:

insert\_values.append('{}.{}.{}'.format(values[index\_number + DATES.index('День:')],

values[index\_number + DATES.index('Месяц:')],

values[index\_number + DATES.index('Год:')]))

else:

insert\_values.append(values[index\_number])

elif (full\_name not in PRIMARY\_KEY\_COLUMNS) and (full\_name not in NO\_INSERT\_COLUMNS):

insert\_values.append(values[index\_number])

return problems, insert\_values

def where\_frame\_input\_check(table\_name, values, column\_names):

date\_count = 0

check\_date = True

problems = []

date\_check = []

integer\_check = []

numeric\_check = []

insert\_values = []

for i in range(10):

date\_check.append(str(i))

integer\_check.append(str(i))

numeric\_check.append(str(i))

integer\_check.append(' ')

numeric\_check.append(' ')

numeric\_check.append('.')

for index in range(len(column\_names)):

index\_number = 2\*date\_count+index

full\_name = table\_name + '.' + column\_names[index]

if full\_name in FOREIGN\_KEY\_COLUMNS:

foreign\_keys = db\_get('SELECT {}\n'

'FROM {}'.format(FOREIGN\_KEY\_TABLE\_COLUMN[full\_name],

'{}.{}'.format(SCHEMA\_NAME, FOREIGN\_KEY\_TABLE[full\_name])))

if values[index\_number] != NULL\_VALUE:

if (values[index\_number] not in foreign\_keys) and (values[index\_number] not in DEFAULT\_VALUES):

problems.append(column\_names[index])

elif full\_name in BOOLEAN\_COLUMNS:

if values[index\_number] not in BOOLEAN\_DEFAULTS:

problems.append(column\_names[index])

elif full\_name in NUMERIC\_COLUMNS:

error\_count = sum(values[index\_number][element] not in numeric\_check

for element in range(len(values[index\_number])))

if error\_count != 0:

problems.append(column\_names[index])

else:

for elem in values[index\_number].split():

if elem.count('.') > 1:

problems.append(column\_names[index])

break

elif full\_name in INTEGER\_COLUMNS:

error\_count = sum(values[index\_number][element] not in integer\_check

for element in range(len(values[index\_number])))

if error\_count != 0:

problems.append(column\_names[index])

elif full\_name in DATE\_COLUMNS:

check\_date\_count = sum(values[index\_number+times] == DEFAULT\_VALUES[0]

for times in range(len(DATES)))

if (check\_date\_count > 0) and (check\_date\_count < 3):

problems.append(column\_names[index])

elif check\_date\_count == 0:

date = []

for times in range(len(DATES)):

error\_count = sum(values[index\_number+times][element] not in date\_check

for element in range(len(values[index\_number+times])))

if error\_count != 0:

problems.append(column\_names[index])

check\_date = False

break

else:

date.append(int(values[index\_number+times]))

if check\_date:

if date[1] not in MONTH\_LIST:

problems.append(column\_names[index])

else:

if type(DAY\_DICT[date[1]]) is list:

if date[2] % 4 == 0:

max\_day = DAY\_DICT[date[1]][1]

else:

max\_day = DAY\_DICT[date[1]][0]

if (date[0] > max\_day) and (date[0] == 0):

problems.append(column\_names[index])

else:

if (date[0] > DAY\_DICT[date[1]]) or (date[0] == 0):

problems.append(column\_names[index])

date\_count += 1

return problems

def where\_function(table\_name, column\_names, values, action\_name):

date\_count = 0

part\_operation = ' WHERE '

filters = []

for index in range(len(column\_names)):

index\_number = 2\*date\_count+index

full\_name = '{}.{}'.format(table\_name, column\_names[index])

if values[2\*date\_count+index] not in DEFAULT\_VALUES:

full\_name = '{}.{}'.format(table\_name, column\_names[index])

if full\_name in FOREIGN\_KEY\_COLUMNS:

if values[index\_number] == NULL\_VALUE:

filters.append('{} is {}'.format(column\_names[index], 'Null'))

else:

if full\_name in INTEGER\_COLUMNS:

filters.append('{} = {}'.format(column\_names[index], values[index\_number][0]))

if full\_name in VARCHAR\_COLUMNS:

filters.append("{} = '{}'".format(column\_names[index], values[index\_number][0]))

elif full\_name in DATE\_COLUMNS:

date = []

if values[index\_number] in DEFAULT\_VALUES:

continue

for element in range(len(DATES)):

if element != 2:

if len(values[index\_number+element]) == 1:

date.append('0{}'.format(values[index\_number+element]))

else:

date.append(values[index\_number+element])

else:

date.append(values[index\_number+element])

filters.append("{} = '{}'".format(column\_names[index], '.'.join(date)))

elif (full\_name in INTEGER\_COLUMNS) or (full\_name in NUMERIC\_COLUMNS):

numbers = values[index\_number].split()

filters.append('{} IN ({})'.format(column\_names[index], ', '.join(numbers)))

elif full\_name in VARCHAR\_COLUMNS:

if action\_name == VIEW\_LIST[0]:

filters.append("{} LIKE '%{}%'".format(column\_names[index], values[index\_number]))

else:

filters.append("{} = '{}'".format(column\_names[index], values[index\_number]))

elif full\_name in BOOLEAN\_COLUMNS:

filters.append('{} = {}'.format(column\_names[index], values[index\_number]))

if full\_name in DATE\_COLUMNS:

date\_count += 1

part\_operation += ' AND '.join(filters)

return part\_operation

def insert\_action(table\_name, column\_names, window\_title, action\_name):

partial\_count = 0

new\_values = ''

frame\_name = FRAME\_NAMES[1].format(action\_name)

insert\_button\_name = ACTION\_LIST[0]

action\_window\_title = '{} ({} данных)'.format(window\_title, ACTION\_LIST[0])

frame\_layout, insert\_column\_names, total\_count = values\_frame(partial\_count, table\_name, column\_names)

insert\_frame = [Gui.Frame(frame\_name, frame\_layout, expand\_x=True)]

insert\_action\_layout = [[Gui.Column([insert\_frame], scrollable=True, expand\_x=True)]]

insert\_button = Gui.Button(button\_text=insert\_button\_name, expand\_x=True)

back\_button = Gui.Button(button\_text='Назад к таблице', expand\_x=True)

insert\_action\_layout.append([insert\_button, back\_button])

insert\_action\_window = Gui.Window(title=action\_window\_title, layout=insert\_action\_layout, modal=True)

while True:

event, values = insert\_action\_window.read()

if event == Gui.WIN\_CLOSED or event != insert\_button\_name:

break

else:

no\_input\_count = sum(values[index] in DEFAULT\_VALUES for index in range(len(values)))

if no\_input\_count == 0:

problem\_columns, insert\_values = values\_frame\_input\_check(partial\_count, table\_name,

values, column\_names)

if len(problem\_columns) == 0:

insert\_operation = INSERT\_TABLE\_EXECUTE.format(SCHEMA\_NAME, table\_name,

','.join(insert\_column\_names), tuple(insert\_values))

db\_edit(insert\_operation)

break

else:

input\_error('', [], frame\_name, problem\_columns)

else:

no\_input\_error(frame\_name)

insert\_action\_window.close()

return db\_get(SHOW\_TABLE\_EXECUTE+table\_name)

def update\_action(table\_name, column\_names, window\_title, action\_name):

new\_values = ''

update\_button\_name = ACTION\_LIST[1]

frame\_name\_1 = FRAME\_NAMES[0].format(action\_name)

frame\_name\_2 = FRAME\_NAMES[1].format(action\_name)

action\_window\_title = '{} ({} данных)'.format(window\_title, ACTION\_LIST[1])

frame\_layout\_1, partial\_count = where\_frame(table\_name, column\_names)

update\_frame\_1 = [Gui.Frame(frame\_name\_1, frame\_layout\_1, expand\_x=True)]

frame\_layout\_2, update\_column\_names, total\_count = values\_frame(partial\_count, table\_name, column\_names)

update\_frame\_2 = [Gui.Frame(frame\_name\_2, frame\_layout\_2, expand\_x=True)]

update\_action\_layout = [[Gui.Column([update\_frame\_1], size=(100, 125), scrollable=True, expand\_x=True)],

[Gui.Column([update\_frame\_2], size=(100, 125), scrollable=True, expand\_x=True)]]

insert\_button = Gui.Button(button\_text=update\_button\_name, expand\_x=True)

back\_button = Gui.Button(button\_text='Назад к таблице', expand\_x=True)

update\_action\_layout.append([insert\_button, back\_button])

insert\_action\_window = Gui.Window(title=action\_window\_title, layout=update\_action\_layout, modal=True)

while True:

event, values = insert\_action\_window.read()

if event == Gui.WIN\_CLOSED or event != update\_button\_name:

break

else:

no\_input\_count\_2 = sum((values[index] not in DEFAULT\_VALUES) and (index >= partial\_count)

for index in range(len(values)))

no\_input\_count\_1 = sum((values[index] not in DEFAULT\_VALUES) and (index < partial\_count)

for index in range(len(values)))

if no\_input\_count\_2 != 0:

problem\_columns\_1 = where\_frame\_input\_check(table\_name, values, column\_names)

problem\_columns\_2, update\_values = values\_frame\_input\_check(partial\_count, table\_name,

values, column\_names)

if (len(problem\_columns\_1) == 0) and (len(problem\_columns\_2) == 0):

update\_where = ''

update\_sets = []

for value\_index in range(len(update\_column\_names)):

if sum(str(update\_values[value\_index]) in elem for elem in DEFAULT\_VALUES) == 0:

if isinstance(update\_values[value\_index], str):

update\_sets.append("{}='{}'".format(update\_column\_names[value\_index],

update\_values[value\_index]))

else:

update\_sets.append("{}='{}'".format(update\_column\_names[value\_index],

update\_values[value\_index]))

if no\_input\_count\_1 != 0:

update\_where = where\_function(table\_name, column\_names, values, action\_name)

update\_operation = UPDATE\_TABLE\_EXECUTE.format(SCHEMA\_NAME, table\_name,

','.join(update\_sets), update\_where)

db\_edit(update\_operation)

break

else:

input\_error(frame\_name\_1, problem\_columns\_1, frame\_name\_2, problem\_columns\_2)

else:

no\_input\_error(frame\_name\_2)

insert\_action\_window.close()

return db\_get(SHOW\_TABLE\_EXECUTE+table\_name)

def delete\_action(table\_name, column\_names, window\_title, action\_name):

new\_values = ''

frame\_name = FRAME\_NAMES[0].format(action\_name)

operation = DELETE\_TABLE\_EXECUTE + table\_name

action\_window\_title = '{} ({} данных)'.format(window\_title, ACTION\_LIST[2])

buttons = []

frame\_layout, total\_count = where\_frame(table\_name, column\_names)

delete\_frame = [Gui.Frame(frame\_name, frame\_layout, expand\_x=True)]

delete\_action\_layout = [[Gui.Column([delete\_frame], scrollable=True, expand\_x=True)]]

for index in range(len(DELETE\_LIST)):

delete\_button = Gui.Button(button\_text=DELETE\_LIST[index], expand\_x=True)

buttons.append(delete\_button)

back\_button = Gui.Button(button\_text='Назад к таблице', key=total\_count, expand\_x=True)

buttons.append(back\_button)

delete\_action\_layout.append([buttons])

delete\_action\_window = Gui.Window(title=action\_window\_title, layout=delete\_action\_layout, modal=True)

while True:

event, values = delete\_action\_window.read()

if event == Gui.WIN\_CLOSED or event == total\_count:

break

elif event == DELETE\_LIST[-1]:

db\_edit(operation)

break

else:

no\_input\_count = sum(values[index] in DEFAULT\_VALUES for index in range(len(values)))

if no\_input\_count < len(values):

problem\_columns = where\_frame\_input\_check(table\_name, values, column\_names)

if len(problem\_columns) == 0:

delete\_operation = operation + where\_function(table\_name, column\_names, values, action\_name)

db\_edit(delete\_operation)

break

else:

input\_error(frame\_name, problem\_columns, '', [])

else:

no\_input\_error(frame\_name)

delete\_action\_window.close()

return db\_get(SHOW\_TABLE\_EXECUTE+table\_name)

def data\_filter(table\_name, column\_names, window\_title, action\_name):

new\_values = ''

frame\_name = FRAME\_NAMES[0].format(action\_name)

operation = SHOW\_TABLE\_EXECUTE + table\_name

view\_window\_title = '{} ({})'.format(window\_title, action\_name)

frame\_layout, elements\_count = where\_frame(table\_name, column\_names)

filter\_frame = [Gui.Frame(frame\_name, frame\_layout, expand\_x=True)]

filter\_layout = [[Gui.Column([filter\_frame], scrollable=True, expand\_x=True)]]

filter\_button = Gui.Button(button\_text='Отфильтровать', key=elements\_count, expand\_x=True)

back\_button = Gui.Button(button\_text='Назад к таблице', key=elements\_count+1, expand\_x=True)

filter\_layout.append([filter\_button, back\_button])

filter\_window = Gui.Window(title=view\_window\_title, layout=filter\_layout, modal=True)

while True:

event, values = filter\_window.read()

if event == Gui.WIN\_CLOSED or event == elements\_count+1:

break

else:

no\_count = sum(values[index] in DEFAULT\_VALUES for index in range(len(values)))

if no\_count < len(values):

problem\_columns = where\_frame\_input\_check(table\_name, values, column\_names)

if len(problem\_columns) == 0:

filter\_operation = operation + where\_function(table\_name, column\_names, values, action\_name)

new\_values = db\_get(filter\_operation)

break

else:

input\_error(frame\_name, problem\_columns, '', [])

else:

no\_input\_error(frame\_name)

filter\_window.close()

return new\_values

def show\_report(table\_name, key, action\_window\_title):

back\_button\_text = 'Назад'

data = []

column\_names\_copy = []

report\_name = [Gui.Text(text=REPORTS\_TYPES[table\_name][key], justification='center')]

report\_layout = [report\_name]

if table\_name == REPORTS\_TABLES[0]:

report\_operation = "SELECT Филиал, COUNT(Товар) AS Количество\n" \

"FROM Авто.Ассортимент\_Филиалов\n" \

"WHERE Наличие=true\n" \

"GROUP BY Филиал\n" \

"ORDER BY Филиал\n"

else:

if key // 2 == 0:

function\_1 = 'SUM'

name\_1 = 'Количество'

else:

function\_1 = 'SUM'

name\_1 = 'Сумма'

if key % 2 == 0:

function\_2 = 'MAX'

name\_2 = 'Максимум'

if table\_name == REPORTS\_TABLES[1]:

main\_element = 'Клиент'

else:

main\_element = 'Поставщик'

else:

function\_2 = 'MIN'

name\_2 = 'Минимум'

main\_element = 'Стоимость'

report\_operation = "WITH tab\_1 AS (\n" \

" SELECT EXTRACT(YEAR FROM Подписано) AS Год, EXTRACT(MONTH FROM Подписано) AS Месяц," \

" Филиал, {}({}) AS {}\n" \

" FROM {}.{}\n" \

" GROUP BY EXTRACT(YEAR FROM Подписано), EXTRACT(MONTH FROM Подписано), Филиал\n" \

" ORDER BY EXTRACT(YEAR FROM Подписано), EXTRACT(MONTH FROM Подписано), Филиал\n" \

"), tab\_2 AS (\n" \

" SELECT Год, Месяц, {}({}) AS {}\n" \

" FROM tab\_1\n" \

" GROUP BY Год, Месяц)\n" \

"SELECT Год, TO\_CHAR (TO\_DATE (Месяц::TEXT, 'MM'), 'Month') AS Месяц, Филиал, {}\n" \

"FROM (" \

" SELECT CAST (tab\_2.Год AS INTEGER), tab\_2.Месяц AS Месяц, tab\_1.Филиал, {}\n" \

" FROM tab\_2 INNER JOIN tab\_1 USING(Год, Месяц)\n" \

" WHERE {}={}\n" \

" ORDER BY Год, Месяц) AS tab\_3".format(function\_1, main\_element, name\_1, SCHEMA\_NAME,

table\_name, function\_2, name\_1, name\_2, name\_1,

name\_1, name\_1, name\_2)

with psycopg2.connect(dbname=DBNAME, user=USER, password=PASSWORD, host=HOST, port=PORT) as connection:

with connection.cursor() as cursor:

cursor.execute(report\_operation)

column\_names = [desc[0] for desc in cursor.description]

get\_data = cursor.fetchall()

for elem in column\_names:

column\_names\_copy.append(elem.replace("\_", " "))

for elem in get\_data:

data.append(list(elem))

data\_report = [Gui.Table(values=data, headings=column\_names\_copy, auto\_size\_columns=True,

justification='center', vertical\_scroll\_only=False, expand\_x=True, expand\_y=True)]

report\_layout.append(data\_report)

exit\_button = [Gui.Button(button\_text=back\_button\_text, expand\_x=True, expand\_y=True)]

report\_layout.append(exit\_button)

reports\_window = Gui.Window(title=action\_window\_title, layout=report\_layout,

element\_justification='center', modal=True).Finalize()

reports\_window.Maximize()

while True:

event, values = reports\_window.read()

if event == Gui.WIN\_CLOSED or event == back\_button\_text:

break

reports\_window.close()

def reports\_select(table\_name, window\_title, action\_name):

back\_button\_text = 'Назад'

action\_window\_title = '{} ({})'.format(window\_title, action\_name)

info\_text\_name = "Выберите необходимый тип отчёта нажатием на соотвествующую кнопку ниже.\n" \

"Для из приложения выхода можете нажать на самую последнюю кнопку."

info\_text = [Gui.Text(text=info\_text\_name, justification='center')]

reports\_select\_layout = [info\_text]

for index in range(len(REPORTS\_TYPES[table\_name])):

new\_button = Gui.Button(button\_text=REPORTS\_TYPES[table\_name][index], key=index,

expand\_x=True, expand\_y=True)

reports\_select\_layout.append([new\_button])

exit\_button = [Gui.Button(button\_text=back\_button\_text, expand\_x=True, expand\_y=True)]

reports\_select\_layout.append(exit\_button)

reports\_window = Gui.Window(title=action\_window\_title, layout=reports\_select\_layout,

element\_justification='center', modal=True).Finalize()

reports\_window.Maximize()

while True:

event, values = reports\_window.read()

if event == Gui.WIN\_CLOSED or event == back\_button\_text:

break

else:

show\_report(table\_name, event, action\_window\_title)

reports\_window.close()

def table\_work(table\_name):

back\_to\_menu\_button\_text = 'Назад в меню'

title\_name = "'{}'".format(table\_name.replace("\_", " "))

window\_title = "{} - Таблица {}".format(TITLE, title\_name)

data = []

main\_table = []

view\_layout = []

table\_layout = []

action\_layout = []

column\_names\_copy = []

with psycopg2.connect(dbname=DBNAME, user=USER, password=PASSWORD, host=HOST, port=PORT) as connection:

with connection.cursor() as cursor:

cursor.execute(SHOW\_TABLE\_EXECUTE+table\_name)

column\_names = [desc[0] for desc in cursor.description]

get\_data = cursor.fetchall()

for elem in column\_names:

column\_names\_copy.append(elem.replace("\_", " "))

for elem in get\_data:

data.append(list(elem))

data\_table = Gui.Table(values=data, headings=column\_names\_copy, auto\_size\_columns=True,

justification='center', vertical\_scroll\_only=False, key=0, expand\_x=True, expand\_y=True)

for index in range(len(ACTION\_LIST)):

action\_layout.append(Gui.Button(button\_text=ACTION\_LIST[index], expand\_x=True))

change\_frame = Gui.Frame("Изменить данные", [action\_layout], expand\_x=True)

for index in range(len(VIEW\_LIST)):

if index != 2:

view\_layout.append(Gui.Button(button\_text=VIEW\_LIST[index], expand\_x=True))

elif table\_name in REPORTS\_TABLES:

view\_layout.append(Gui.Button(button\_text=VIEW\_LIST[index], expand\_x=True))

view\_frame = Gui.Frame("Просмотр", [view\_layout], expand\_x=True)

back\_button = Gui.Button(button\_text=back\_to\_menu\_button\_text, expand\_x=True)

table\_layout = [[data\_table], [change\_frame], [view\_frame], [back\_button]]

tables\_window = Gui.Window(title=window\_title, layout=table\_layout,

element\_justification='center', modal=True).Finalize()

tables\_window.Maximize()

while True:

event, values = tables\_window.read()

if event == Gui.WIN\_CLOSED or event == back\_to\_menu\_button\_text:

break

elif event in VIEW\_LIST:

if event == VIEW\_LIST[0]:

new\_data = data\_filter(table\_name, column\_names, window\_title, event)

if new\_data != DEFAULT\_VALUES[0]:

tables\_window[0].update(new\_data)

elif event == VIEW\_LIST[1]:

tables\_window[0].update(db\_get(SHOW\_TABLE\_EXECUTE+table\_name))

else:

reports\_select(table\_name, window\_title, event)

else:

if event == ACTION\_LIST[0]:

tables\_window[0].update(insert\_action(table\_name, column\_names, window\_title, event))

elif event == ACTION\_LIST[1]:

tables\_window[0].update(update\_action(table\_name, column\_names, window\_title, event))

else:

tables\_window[0].update(delete\_action(table\_name, column\_names, window\_title, event))

tables\_window.close()

def table\_select():

exit\_button\_text = 'Выход'

window\_title = "{} - Выбор таблицы".format(TITLE)

info\_text\_name = "Выберите необходимую для работы таблицы нажатием на соотвествующую кнопку ниже.\n" \

"Для из приложения выхода можете нажать на самую последнюю кнопку."

info\_text = [Gui.Text(text=info\_text\_name, justification='center')]

table\_select\_layout = [info\_text]

for index in range(len(TABLES\_LIST)):

button\_text = TABLES\_LIST[index].replace("\_", " ")

new\_button = Gui.Button(button\_text=button\_text, key=index, expand\_x=True, expand\_y=True)

table\_select\_layout.append([new\_button])

exit\_button = [Gui.Button(button\_text=exit\_button\_text, expand\_x=True, expand\_y=True)]

table\_select\_layout.append(exit\_button)

table\_select\_window = Gui.Window(title=window\_title, layout=table\_select\_layout,

element\_justification='center', modal=True).Finalize()

table\_select\_window.Maximize()

while True:

event, values = table\_select\_window.read()

if event == Gui.WIN\_CLOSED or event == exit\_button\_text:

break

else:

table\_work(TABLES\_LIST[event])

table\_select\_window.close()

if \_\_name\_\_ == "\_\_main\_\_":

Gui.set\_options(font=(STYLE, ELEMENT\_SIZE))

Gui.theme('DarkBrown5')

table\_select()

### ПРИЛОЖЕНИЕ Б

Запросы для создания базы данных:

CREATE SCHEMA Авто;

CREATE TABLE Авто.Клиенты(

Номер\_клиента serial PRIMARY KEY,

ФИО varchar,

Город varchar,

Адрес varchar,

Телефон varchar

);

CREATE TABLE Авто.Филиалы(

Номер\_филиала serial PRIMARY KEY,

Город varchar,

Адрес varchar,

Телефон varchar

);

CREATE TABLE Авто.Поставщики(

Номер\_поставщика serial PRIMARY KEY,

Наименование varchar,

Город varchar,

Адрес varchar,

Телефон varchar,

Рейтинг integer

);

CREATE TABLE Авто.Наименования\_Товаров(

Код\_имени serial PRIMARY KEY,

Фирма varchar,

Модель varchar

);

CREATE TABLE Авто.Товары(

Код\_товара varchar PRIMARY KEY,

Наименование serial REFERENCES Авто.Наименования\_Товаров(Код\_имени),

Цвет varchar,

Изготовлено date,

Текущая\_цена numeric DEFAULT 0

);

CREATE TABLE Авто.Должности(

Номер\_должности serial PRIMARY KEY,

Название\_должности varchar,

Оклад numeric DEFAULT 0

);

CREATE TABLE Авто.Сотрудники(

Табельный\_номер varchar PRIMARY KEY,

ФИО varchar,

Дата\_рождения date,

Пол varchar(1),

Телефон varchar,

Паспортные\_данные varchar,

ИНН varchar(12),

Должность serial REFERENCES Авто.Должности(Номер\_должности)

);

CREATE TABLE Авто.Ассортимент\_Филиалов(

Филиал serial REFERENCES Авто.Филиалы(Номер\_филиала),

Товар varchar REFERENCES Авто.Товары(Код\_товара),

Наличие bool,

UNIQUE(Филиал, Товар)

);

CREATE TABLE Авто.Продажи(

Номер\_продажи serial PRIMARY KEY,

Товар varchar REFERENCES Авто.Товары(Код\_товара),

Филиал serial REFERENCES Авто.Филиалы(Номер\_филиала),

Клиент serial REFERENCES Авто.Клиенты(Номер\_клиента),

Подписано date,

Доставлено date CHECK (Доставлено >= Подписано),

Менеджер varchar REFERENCES Авто.Сотрудники(Табельный\_номер),

Стоимость numeric DEFAULT 0

);

CREATE TABLE Авто.Поставки(

Номер\_поставки serial PRIMARY KEY,

Товар varchar REFERENCES Авто.Товары(Код\_товара),

Филиал serial REFERENCES Авто.Филиалы(Номер\_филиала),

Поставщик serial REFERENCES Авто.Поставщики(Номер\_поставщика),

Подписано date,

Доставлено date CHECK (Доставлено >= Подписано),

Менеджер varchar REFERENCES Авто.Сотрудники(Табельный\_номер),

Стоимость numeric DEFAULT 0

);

Запросы для удаления ограничений на значение «NULL» в нужных таблицах:

ALTER TABLE Авто.Продажи ALTER COLUMN Филиал DROP NOT NULL;

ALTER TABLE Авто.Продажи ALTER COLUMN Клиент DROP NOT NULL;

ALTER TABLE Авто.Поставки ALTER COLUMN Филиал DROP NOT NULL;

ALTER TABLE Авто.Поставки ALTER COLUMN Поставщик DROP NOT NULL;

Запросы для создания триггеров и соответствующими им функциями:

CREATE OR REPLACE FUNCTION Авто.before\_insert\_sales\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT Наличие FROM Авто.Ассортимент\_Филиалов WHERE Товар = NEW.Товар AND Филиал = NEW.Филиал) = true THEN

RETURN NEW;

ELSE

RETURN null;

END IF;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER insert\_sales\_trigger

BEFORE INSERT ON Авто.Продажи

FOR EACH ROW

EXECUTE FUNCTION Авто.before\_insert\_sales\_function();

CREATE OR REPLACE FUNCTION Авто.after\_insert\_sales\_function()

RETURNS TRIGGER AS

$$

DECLARE

current\_price numeric;

BEGIN

current\_price = (

SELECT Текущая\_цена

FROM Авто.Товары

WHERE Код\_товара = NEW.Товар

);

UPDATE Авто.Продажи

SET Стоимость = current\_price

WHERE Номер\_продажи = NEW.Номер\_продажи;

UPDATE Авто.Ассортимент\_Филиалов

SET Наличие = '0'

WHERE Товар = NEW.Товар and Филиал = NEW.Филиал;

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER after\_insert\_sales\_trigger

AFTER INSERT ON Авто.Продажи

FOR EACH ROW

EXECUTE FUNCTION Авто.after\_insert\_sales\_function();

CREATE OR REPLACE FUNCTION Авто.before\_insert\_deliveries\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT Наличие FROM Авто.Ассортимент\_Филиалов WHERE Товар = NEW.Товар and Филиал = NEW.Филиал) = false THEN

UPDATE Авто.Ассортимент\_Филиалов

SET Наличие = '1'

WHERE Товар = NEW.Товар and Филиал = NEW.Филиал;

RETURN NEW;

ELSE

RETURN NULL;

END IF;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER before\_insert\_deliveries\_trigger

BEFORE INSERT ON Авто.Поставки

FOR EACH ROW

EXECUTE FUNCTION Авто.before\_insert\_deliveries\_function();

CREATE OR REPLACE FUNCTION Авто.update\_delete\_products\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(1) FROM Авто.Ассортимент\_Филиалов WHERE Ассортимент\_Филиалов.Товар=OLD.Код\_товара)<>0 THEN

DELETE FROM Авто.Ассортимент\_Филиалов WHERE Ассортимент\_Филиалов.Товар=OLD.Код\_товара;

END IF;

IF (SELECT COUNT(Номер\_продажи) FROM Авто.Продажи WHERE Продажи.Товар=OLD.Код\_товара)<>0 THEN

UPDATE Авто.Продажи

SET Товар=NULL

WHERE Продажи.Товар=OLD.Код\_товара;

END IF;

IF (SELECT COUNT(Номер\_поставки) FROM Авто.Поставки WHERE Поставки.Менеджер=OLD.Код\_товара)<>0 THEN

UPDATE Авто.Поставки

SET Товар=NULL

WHERE Поставки.Товар=OLD.Код\_товара;

END IF;

IF TG\_OP = 'UPDATE' THEN

RETURN NEW;

ELSE

RETURN OLD;

END IF;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER update\_delete\_products\_trigger

BEFORE UPDATE OR DELETE ON Авто.Товары

FOR EACH ROW

EXECUTE FUNCTION Авто.update\_delete\_products\_function();

CREATE OR REPLACE FUNCTION Авто.update\_delete\_workers\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(Номер\_продажи) FROM Авто.Продажи WHERE Продажи.Менеджер=OLD.Табельный\_номер)<>0 THEN

UPDATE Авто.Продажи

SET Менеджер=NULL

WHERE Продажи.Менеджер=OLD.Табельный\_номер;

END IF;

IF (SELECT COUNT(Номер\_поставки) FROM Авто.Поставки WHERE Поставки.Менеджер=OLD.Табельный\_номер)<>0 THEN

UPDATE Авто.Поставки

SET Менеджер=NULL

WHERE Поставки.Менеджер=OLD.Табельный\_номер;

END IF;

IF TG\_OP = 'UPDATE' THEN

RETURN NEW;

ELSE

RETURN OLD;

END IF;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER update\_delete\_workers\_trigger

BEFORE UPDATE OR DELETE ON Авто.Сотрудники

FOR EACH ROW

EXECUTE FUNCTION Авто.update\_delete\_workers\_function();

CREATE OR REPLACE FUNCTION Авто.delete\_clients\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(Номер\_продажи) FROM Авто.Продажи WHERE Продажи.Клиент=OLD.Номер\_клиента)<>0 THEN

UPDATE Авто.Продажи

SET Клиент=Null

WHERE Продажи.Клиент=OLD.Номер\_клиента;

END IF;

RETURN OLD;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER delete\_clients\_trigger

BEFORE DELETE ON Авто.Клиенты

FOR EACH ROW

EXECUTE FUNCTION Авто.delete\_clients\_function();

CREATE OR REPLACE FUNCTION Авто.delete\_branches\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(1) FROM Авто.Ассортимент\_Филиалов WHERE Ассортимент\_Филиалов.Филиал=OLD.Номер\_филиала)<>0 THEN

DELETE FROM Авто.Ассортимент\_Филиалов WHERE Ассортимент\_Филиалов.Филиал=OLD.Номер\_филиала;

END IF;

IF (SELECT COUNT(Номер\_продажи) FROM Авто.Продажи WHERE Продажи.Филиал=OLD.Номер\_филиала)<>0 THEN

UPDATE Авто.Продажи

SET Филиал=Null

WHERE Продажи.Филиал=OLD.Номер\_филиала;

END IF;

IF (SELECT COUNT(Номер\_поставки) FROM Авто.Поставки WHERE Поставки.Филиал=OLD.Номер\_филиала)<>0 THEN

UPDATE Авто.Поставки

SET Филиал=Null

WHERE Поставки.Филиал=OLD.Номер\_филиала;

END IF;

RETURN OLD;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER delete\_braches\_trigger

BEFORE DELETE ON Авто.Филиалы

FOR EACH ROW

EXECUTE FUNCTION Авто.delete\_branches\_function();

CREATE OR REPLACE FUNCTION Авто.delete\_providers\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(Номер\_поставки) FROM Авто.Поставки WHERE Поставки.Поставщик=OLD.Номер\_поставщика)<>0 THEN

UPDATE Авто.Поставки

SET Поставщик=Null

WHERE Поставки.Поставщик=OLD.Номер\_поставщика;

END IF;

RETURN OLD;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER delete\_providers\_trigger

BEFORE DELETE ON Авто.Поставщики

FOR EACH ROW

EXECUTE FUNCTION Авто.delete\_providers\_function();

CREATE OR REPLACE FUNCTION Авто.delete\_products\_names\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(Код\_товара) FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени)<>0 THEN

IF (SELECT COUNT(1) FROM Авто.Ассортимент\_Филиалов WHERE Ассортимент\_Филиалов.Товар IN

(SELECT Код\_товара FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени))<>0 THEN

DELETE FROM Авто.Ассортимент\_Филиалов WHERE Ассортимент\_Филиалов.Товар IN

(SELECT Код\_товара FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени);

END IF;

IF (SELECT COUNT(Номер\_продажи) FROM Авто.Продажи WHERE Продажи.Товар IN

(SELECT Код\_товара FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени))<>0 THEN

UPDATE Авто.Продажи

SET Товар=Null

WHERE Продажи.Товар IN

(SELECT Код\_товара FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени);

END IF;

IF (SELECT COUNT(Номер\_поставки) FROM Авто.Поставки WHERE Поставки.Товар IN

(SELECT Код\_товара FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени))<>0 THEN

UPDATE Авто.Поставки

SET Товар=Null

WHERE Поставки.Товар IN

(SELECT Код\_товара FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени);

END IF;

DELETE FROM Авто.Товары WHERE Товары.Наименование=OLD.Код\_имени;

END IF;

RETURN OLD;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER delete\_products\_names\_trigger

BEFORE DELETE ON Авто.Наименования\_Товаров

FOR EACH ROW

EXECUTE FUNCTION Авто.delete\_products\_names\_function();

CREATE OR REPLACE FUNCTION Авто.delete\_posts\_function()

RETURNS TRIGGER AS

$$

BEGIN

IF (SELECT COUNT(Табельный\_номер) FROM Авто.Сотрудники WHERE Сотрудники.Должность=OLD.Номер\_должности)<>0 THEN

IF (SELECT COUNT(Номер\_продажи) FROM Авто.Продажи WHERE Продажи.Менеджер IN

(SELECT Табельный\_номер FROM Авто.Сотрудники WHERE Сотрудники.Должность=OLD.Номер\_должности))<>0 THEN

UPDATE Авто.Продажи

SET Менеджер=Null

WHERE Продажи.Менеджер IN

(SELECT Табельный\_номер FROM Авто.Сотрудники WHERE Сотрудники.Должность=OLD.Номер\_должности);

END IF;

IF (SELECT COUNT(Номер\_поставки) FROM Авто.Поставки WHERE Поставки.Менеджер IN

(SELECT Табельный\_номер FROM Авто.Сотрудники WHERE Сотрудники.Должность=OLD.Номер\_должности))<>0 THEN

UPDATE Авто.Поставки

SET Менеджер=Null

WHERE Поставки.Менеджер IN

(SELECT Табельный\_номер FROM Авто.Сотрудники WHERE Сотрудники.Должность=OLD.Номер\_должности);

END IF;

DELETE FROM Авто.Сотрудники WHERE Сотрудники.Должность=OLD.Номер\_должности;

END IF;

RETURN OLD;

END;

$$

LANGUAGE 'plpgsql';

CREATE TRIGGER delete\_posts\_trigger

BEFORE DELETE ON Авто.Должности

FOR EACH ROW

EXECUTE FUNCTION Авто.delete\_posts\_function();