



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение высшего
образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО МГТУ «СТАНКИН»)


Институт информационных технологий


Кафедра информационных систем

Проект по дисциплине «Управление данными» на тему:
Проектирование БД ветеринарной клиники

Студент
группа ИДБ-22-06

Руководитель
старший преподаватель



подпись


подпись

Чausова Д.Д.

Быстрикова В.А.

Москва 2024 г.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
1. ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ	4
1.1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	4
1.2. КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ	13
1.3. ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ	18
1.4. ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ	26
2. ОПИСАНИЕ ФУНКЦИОНИРОВАНИЯ БД	35
2.1. НАЗНАЧЕНИЕ И ПЕРЕЧЕНЬ ФУНКЦИЙ БАЗЫ ДАННЫХ	35
2.2. ОПИСАНИЕ РАБОТЫ С БАЗОЙ ДАННЫХ	37
ЗАКЛЮЧЕНИЕ	66
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	68
ПРИЛОЖЕНИЕ А. ЗАПОЛНЕНИЕ ТАБЛИЦ ДАННЫМИ	69
ПРИЛОЖЕНИЕ Б. SQL-КОМАНДЫ СОЗДАНИЯ ПРОГРАММНЫХ ОБЪЕКТОВ БД	76
ПРИЛОЖЕНИЕ В. КОД ПРОГРАММЫ	101

ВВЕДЕНИЕ

Цель проекта – изучение и создание базы данных для ветеринарной клиники.

В настоящее время ветеринарные клиники сталкиваются с растущим объемом данных, поступающих из различных источников, таких как записи о пациентах, история их болезней, результаты анализов, графики вакцинации, данные о проведенных процедурах и информация о владельцах животных. Для эффективной работы и обслуживания клиентов необходимо внедрение современных технологий, а база данных является ключевым элементом в этом процессе.

Проектирование базы данных для ветеринарной клиники может решить ряд проблем и вызовов, с которыми сталкиваются ветеринарные учреждения. Включение в базу данных инструментов для анализа данных, например, хранение данных в удобном для анализа формате и поддержка запросов для извлечения нужной информации, позволит оптимизировать ведение истории болезни пациентов, планирование профилактических осмотров, управление складскими запасами и многое другое.

Для выполнения проекта необходимо провести анализ предметной области и определить функции ветеринарной клиники.

Затем следует логическое проектирование, включающее ER-диаграммы для показа связей между сущностями. Физическое проектирование включает определение организации файлов, индексов и ограничений для обеспечения эффективного доступа к данным. После этого требуется создать интерфейс для работы с базой данных.

Таким образом, целью проекта является изучение и создание базы данных для ветеринарной клиники, чтобы обеспечить эффективное хранение и управление информацией о клиентах, врачах, услугах, лекарственных препаратах и других данных, необходимых для качественного ведения медицинских записей животных и обслуживания клиентов.

1. ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БАЗЫ ДАННЫХ

1.1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Предметную область можно определить как сферу человеческой деятельности, выделенную и описанную согласно установленным критериям. В описываемое понятие должны входить сведения об ее элементах, явлениях, отношениях и процессах, отражающих различные аспекты этой деятельности.

Одна из первых задач, с решением которых сталкивается разработчик программной системы – это изучение, осмысление и анализ предметной области. Дело в том, что предметная область сильно влияет на все аспекты проекта: требования к системе, взаимодействие с пользователем, модель хранения данных, реализацию и т.д.

Анализ предметной области позволяет выделить ее сущности, определить первоначальные требования к функциональности и определить границы проекта.

В данной работе в качестве предметной области рассматривается деятельность ветеринарной клиники.

Актуальность данной предметной области заключается в том, что забота о здоровье и благополучии наших домашних питомцев становится все более важной в современном мире. Ветеринарная клиника – это не просто место оказания профессиональной помощи животным, это место, где сочетаются забота, внимание и высокие технологии в медицинском обслуживании. В сегодняшней суете и стрессовом образе жизни люди все больше обращают внимание на своих четвероногих друзей как на членов семьи. Ответственность за заботу о здоровье и комфорте животных приводит к повышенному интересу к услугам ветеринарной медицины.

Ветеринарные клиники становятся неотъемлемой частью поддержания здоровья, диагностики и лечения животных, обеспечивая им качественное и профессиональное медицинское обслуживание. Благодаря постоянному развитию технологий и научных открытий в ветеринарной медицине,

современные клиники предлагают широкий спектр услуг – от профилактических осмотров и вакцинации до сложных операций и реабилитации животных. Основные задачи, решаемые в данной предметной области, включают:

- учёт посещений и процедур: возможность записи пациентов на прием, предоставление лечения и процедур;
- контроль лекарственных препаратов и оборудования: ведение учета запасов лекарств, контроль сроков годности, оформление накладных и инвентаризация;
- доступ к данным питомцев и их владельцев;
- формирование электронной медицинской карты питомца: добавление первичных осмотров, вакцинаций, анализов, данных инструментальных исследований и т.д.;
- работа с расписанием приема специалистов, работы кабинетов, направление питомцев на прием к специалистам, исследования,
- ведение взаиморасчетов с клиентами, выписка счетов, прием оплат, ведение лицевых счетов,
- ведение отчетной деятельности, обработка звонков.

Существует множество программных продуктов, решающих задачи в данной области:

- программный продукт VetMaster, VetSoft;
- интегрированная система учета для ветеринарных клиник Ветеринарное Поле, ВетИнфо.ру., Vetport.;
- конфигурация Ветеринарная клиника;
- программный продукт для учета и планирования ветеринарных процедур (Ветис, ВетДоктор) и другие.

В качестве первого программного продукта рассмотрим сайт VetMaster [2]. VetMaster – программный продукт для автоматизации ветеринарных клиник, учета пациентов и медицинской истории.

Портал для подбора для автоматизации ветеринарных клиник, учета пациентов и медицинской истории VetMaster предоставляет следующие основные возможности:

- Онлайн-запись (рис.1.1). Пациенты могут записываться на прием или процедуры через онлайн-сервис, что упрощает процесс планирования и облегчает коммуникацию с клиникой.

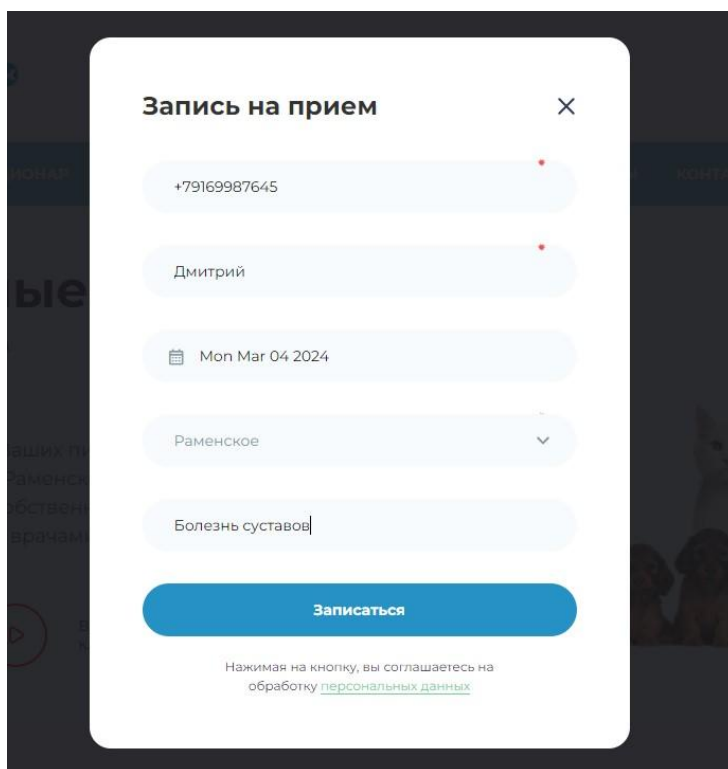


Рис 1.1 Онлайн-запись

- Просмотр расписания врачей и выбор подходящего времени приёма.
- Покупка ветеринарных препаратов и кормов для животных.
- Электронная медицинская карта питомца: позволяет владельцам животных создавать и вести электронные медицинские карты своих питомцев с историей болезни, процедурами и т.д, которые хранятся в базе данных клиники.
- Управление складом – включает возможность учета и управления запасами лекарств, инструментов и других материалов клиники.
- Регистрация и прикрепление новых клиентов и их питомцев, где хранится информация об виде, породе, весе, росте и т.д. животного.

- Просмотр услуг (рис.1.2) – возможность выбрать подходящие процедуры, операции, вакцинации и другие мероприятия, а также планировать расписание на каждый день.

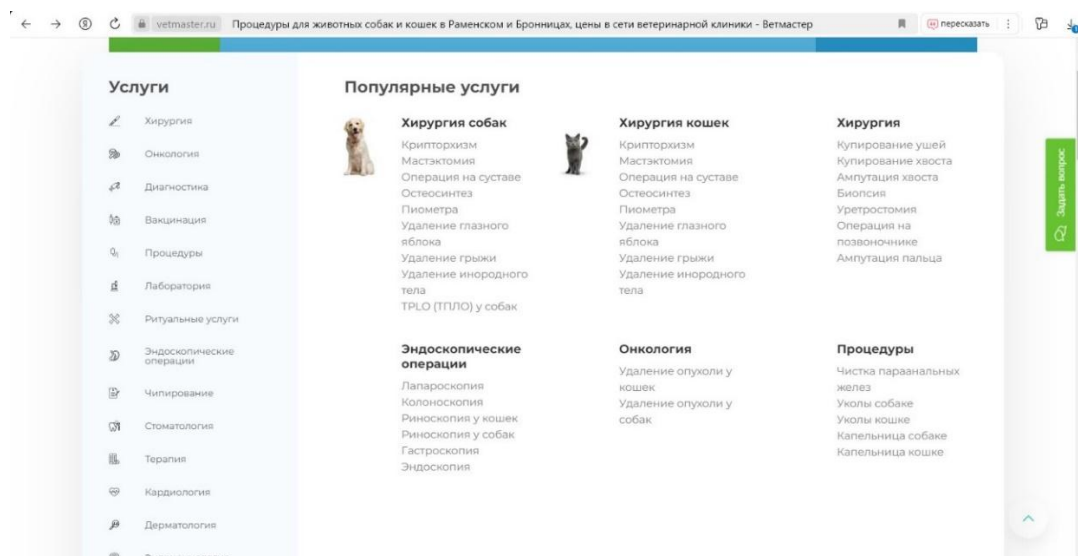


Рис. 1.2 Просмотр услуг

- Просмотр стоимости услуг (рис 1.3), включающий возможность узнать цены на многочисленные процедуры, в которых нуждается питомец.

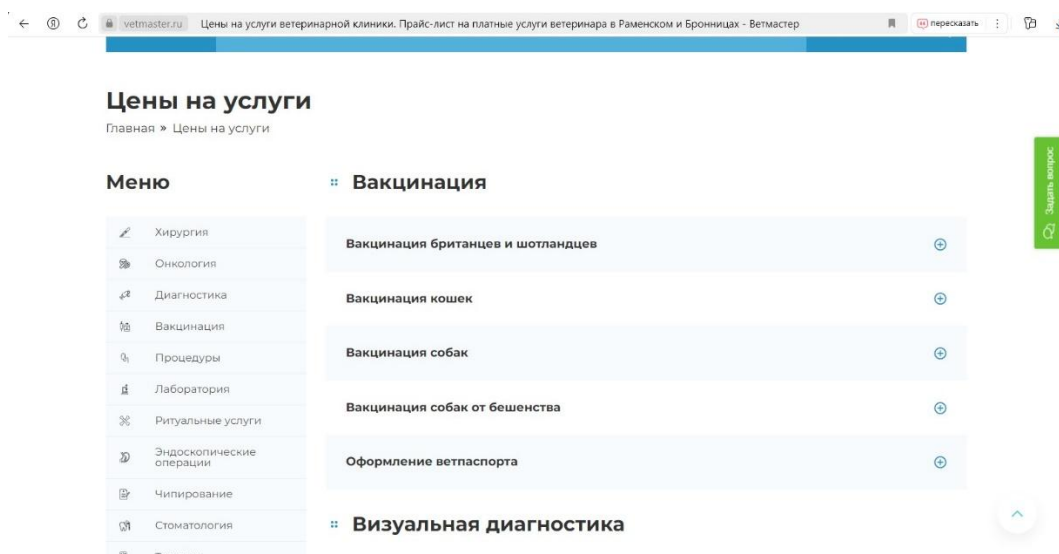


Рис. 1.3 Просмотр стоимости услуг

В качестве второго программного продукта рассмотрим конфигурацию «Ветеринарная клиника» [1]. Конфигурация "Ветеринарная клиника" предназначена для автоматизации работы ветеринарных клиник. Данная программа позволяет усовершенствовать процесс обслуживания животных-пациентов и их хозяев благодаря ведению всестороннего учета. Имеется

возможность учета визитов клиентов, оказанных услуг, вакцинаций, складской учет (приход и продажа товаров, состояние склада) а также получение отчетов по этим данным. Хранение информации о животных-пациентах, их хозяевах и всех их посещениях, а также хранение истории лечения животных.

Среди основных функций данной программы стоит выделить:

- Учёт врачей и специализаций (рис.1.4), который позволяет добавлять информацию о врачах и их специализациях, что помогает в управлении персоналом клиники и оказании специализированной помощи питомцам.

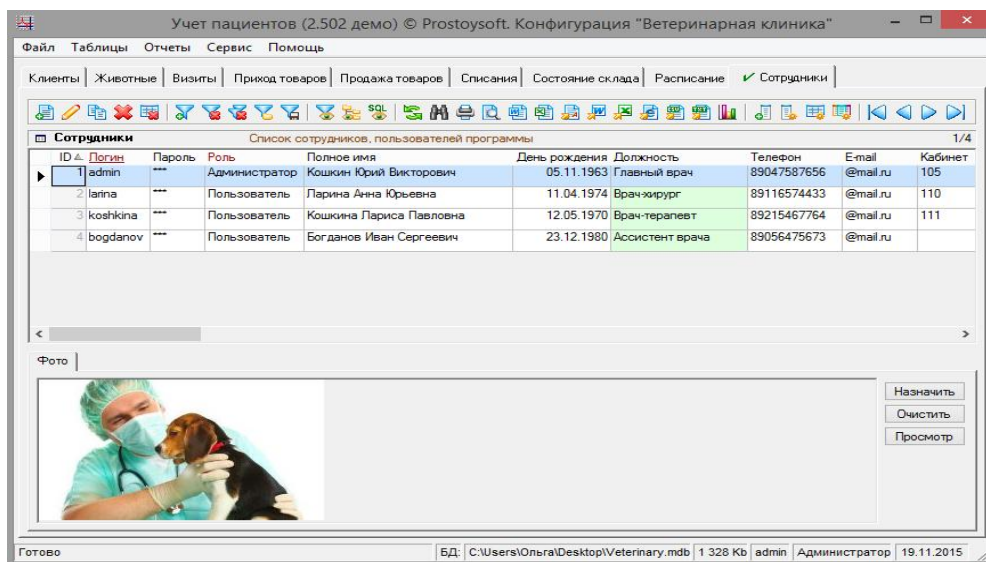


Рис.1.4 Учёт врачей и специализаций

- Учёт животных-пациентов клиники (рис.1.5) предполагает ведение списка животных-пациентов с полной информацией по ним. По каждому животному можно просмотреть историю его лечения, обследования и поставленные диагнозы.

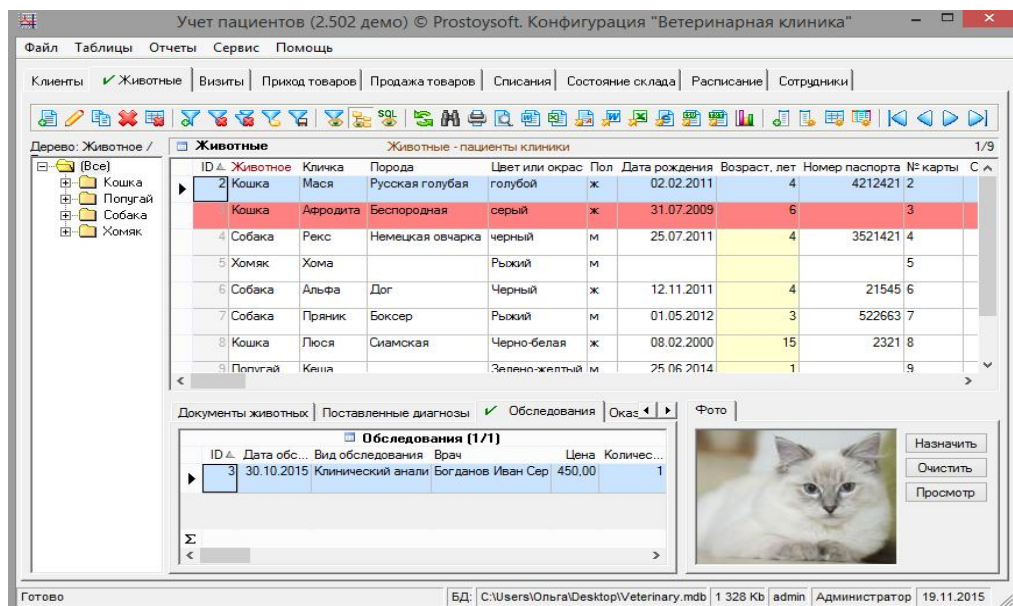


Рис.1.5 Учёт животных-пациентов клиники

- Учёт клиентов-владельцев животных (рис.1.6). Ведение списка клиентов-владельцев животных и их контактной информации. По каждому клиенту можно посмотреть полную информацию о его питомцах-пациентах ветеринарной клиники.

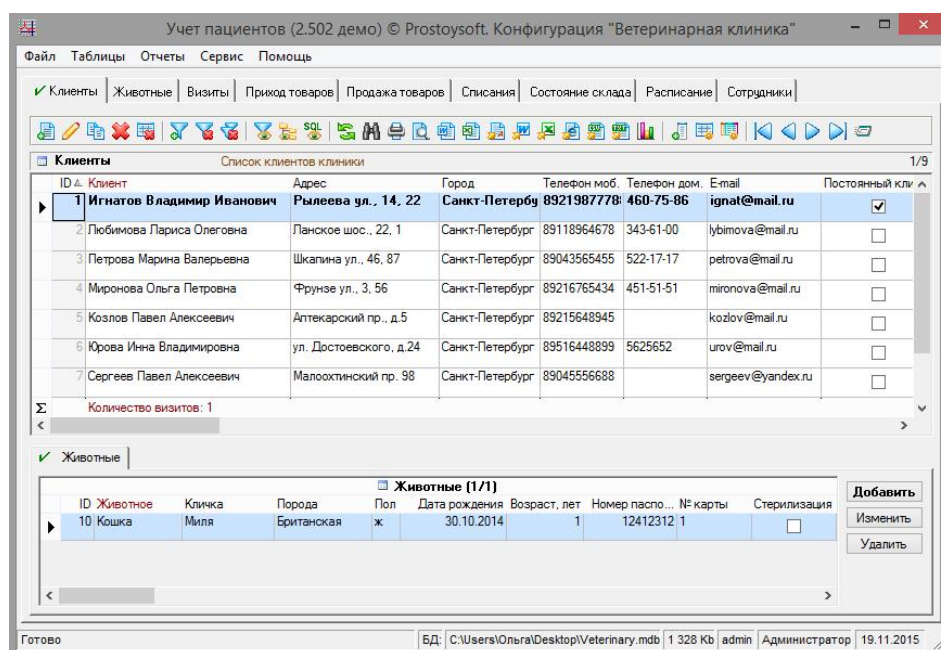


Рис.1.6 Учёт клиентов-владельцев животных

- Учёт визитов и оказанных услуг (рис.1.7) предоставляет возможность фиксировать медицинские процедуры, лекарства, диагностику и другие услуги, которые были предоставлены питомцам.

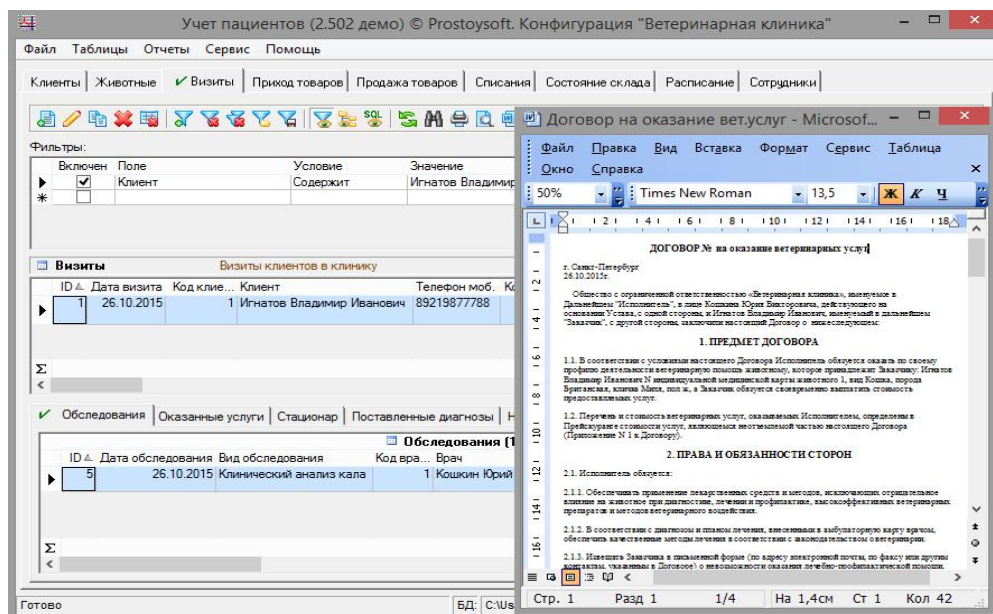


Рис.1.7 Учёт визитов и оказанных услуг

- Финансовый учёт: Помогает вести финансовый учет клиники, включая формирование счетов, учет оплаты услуг, анализ доходов и расходов.
- Составление расписания работы персонала: Ведение расписания дежурств сотрудников ветеринарной клиники, предварительная запись клиентов на прием.
- Складской учёт (рис.1.8). Учет основных складских операций – поступление и продажа товаров, списание расходных средств. Учет остатков на складе.
- Формирование отчётов: Формирование отчетов по оказанным услугам, ежедневным визитам пациентов в клинику состоянию склада и т.д. Возможность создавать новые отчеты.

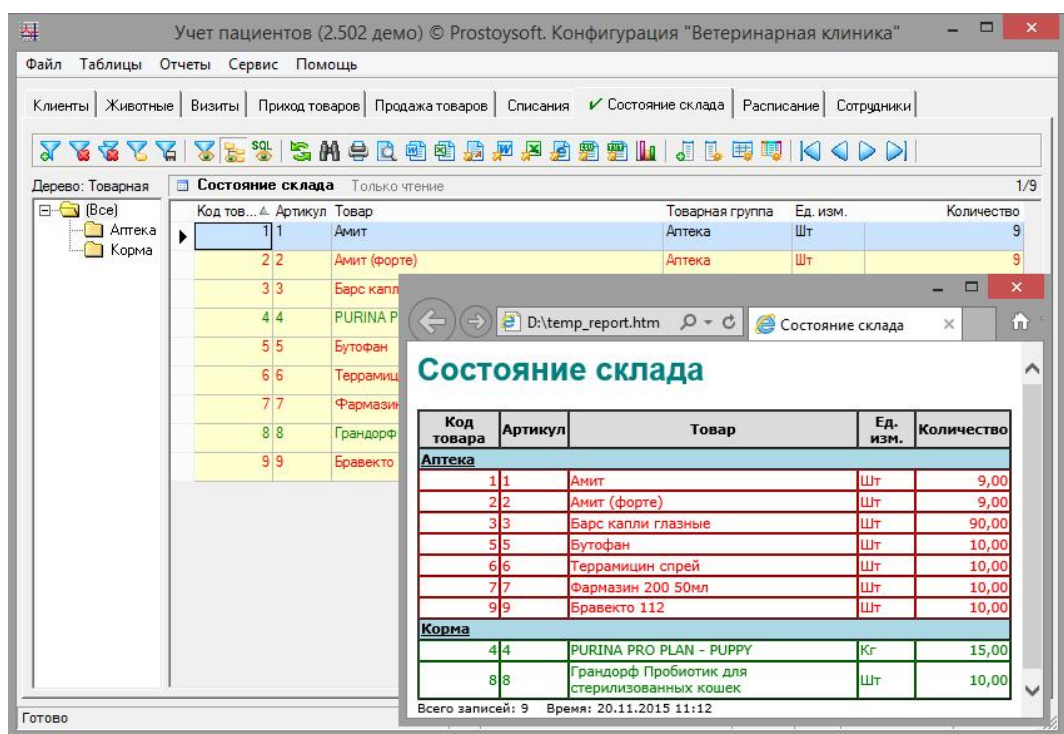


Рис.1.8 Складской учёт

Для рассмотренных программных продуктов можно выделить общие функции:

- Учёт и хранение данных о животных.
- Планирование и учёт приёмов.
- Управление складом.
- Управление услугами.

Основной особенностью портала для подбора для автоматизации ветеринарных клиник, учета пациентов и медицинской истории VetMaster может заключаться в широком спектре ветеринарных услуг, таких как медицинское обслуживание животных, хирургические вмешательства, диагностика, стоматология, профилактика заболеваний и другие аспекты, связанные с заботой о здоровье животных. Кроме того, на сайте могут быть представлены консультации ветеринарных специалистов, информация о клиниках и контактная информация для связи.

Стоит отметить и конфигурацию «Ветеринарная клиника», которая представляет собой программное обеспечение, специально разработанное для автоматизации работы ветеринарных клиник. Особенности этой

конфигурации тоже включают в себя: учёт медицинских карт пациентов, управление расписанием приемов, удобный доступ к данным и поддержка отчетности и аналитики. Программа предоставляет простой и удобный доступ к данным о пациентах, что позволяет оперативно находить и анализировать необходимую информацию.

После проведения анализа предметной области был выделен перечень функций, которые будут реализованы в данной работе:

- Учёт животных и клиентов-владельцев.
- Запись на приём и управление расписанием работы персонала.
- Добавление функционала для ведения учёта и истории визитов питомцев в ветеринарную клинику.
- Создание и ведение электронных медицинских карт животных.
- Управление услугами и процедурами.
- Расход и закупка медицинских препаратов и кормов для животных.

1.2. КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Концептуальное проектирование заключается в формализованном описании предметной области, при этом данное описание не должно зависеть от конкретной СУБД. Концептуальное проектирование отражает обобщённую модель предметной области, для которой создается в БД. Цель данного типа заключается в определении содержания БД.

Концептуальное проектирование задаётся следующим вопросом: «Какие данные должны храниться в базе данных?». Ответом на данный вопрос следует из анализа задач и хранимых объектов.

Чтобы провести анализ задач используется диаграмма вариантов использования или иначе – диаграмма прецедентов. Данная диаграмма отражает взаимодействие действующего лица с вариантами использования системы, где действующее лицо – это сущность, объект, взаимодействующая с моделируемой системой, а вариант использования – это набор функций, предоставляющая моделируемая система действующему лицу.

Диаграмма прецедентов лишь описывает сам принцип взаимодействия, не указывая при этом сам подход к реализации.

Анализ предметной области непосредственно связан с концептуальным проектированием путём того, что результатом анализа служит перечень функций, которые будут реализованы в системе. А концептуальное проектирование в свою очередь описывает взаимодействие и отношение данных функций с различными действующими лицами.

В проектируемой системе для ветеринарной клиники реализованы разные функции в зависимости от роли сотрудника. Это связано с тем, что каждое действующее лицо выполняет определенные задачи, которые помогают обеспечить эффективность работы всей клиники.

Обязанности администратора подразумевают учёт животных, клиентов-владельцев, запись на приём и управление расписанием персонала, а также ведение электронных медицинских карт животных, так как

администратор является ключевой фигурой, отвечающей за общую организацию и координацию работы клиники.

Медицинский статистик отвечает за добавление функционала для ведения учёта и истории визитов питомцев, а также управление услугами и процедурами. Это связано с необходимостью систематизации и структурирования информации о пациентах и оказываемых услуг. Статистик играет важную роль в обеспечении прозрачности и эффективности работы клиники.

Менеджер по закупкам, в свою очередь, отвечает за учёт расхода медицинских препаратов и кормов, а также за их закупку. Это отдельная функция, требующая специальных знаний и навыков управления запасами и финансами.

Ветеринарный врач отвечает за создание и ведение электронных медицинских карт животных, а также за назначение услуг и выполнение процедур. Это его основная профессиональная деятельность, требующая медицинских знаний и навыков. Ветеринарный врач является основным исполнителем лечебных и диагностических мероприятий, поэтому его обязанности чётко отделены от административных.

Таким образом, в данной системе прослеживается чёткое разграничение обязанностей между различными ролями, что позволяет обеспечить эффективное управление, учёт, контроль и оказание необходимых услуг. Совпадение некоторых обязанностей (например, ведение электронных медицинских карт) объясняется необходимостью взаимодействия и координации между специалистами для обеспечения целостности и непрерывности процесса оказания ветеринарной помощи.

Каждое действующее лицо выполняет свою четко определенную роль в рамках общей системы работы клиники. Каждая роль важна для эффективного функционирования всех процессов ветеринарной клиники. Нельзя, например, допустить, чтобы администратор выполнял функции врача, поскольку у него нет соответствующего образования и опыта.

Диаграмма вариантов использования ветеринарной клиники представлена на рис. 1.9.

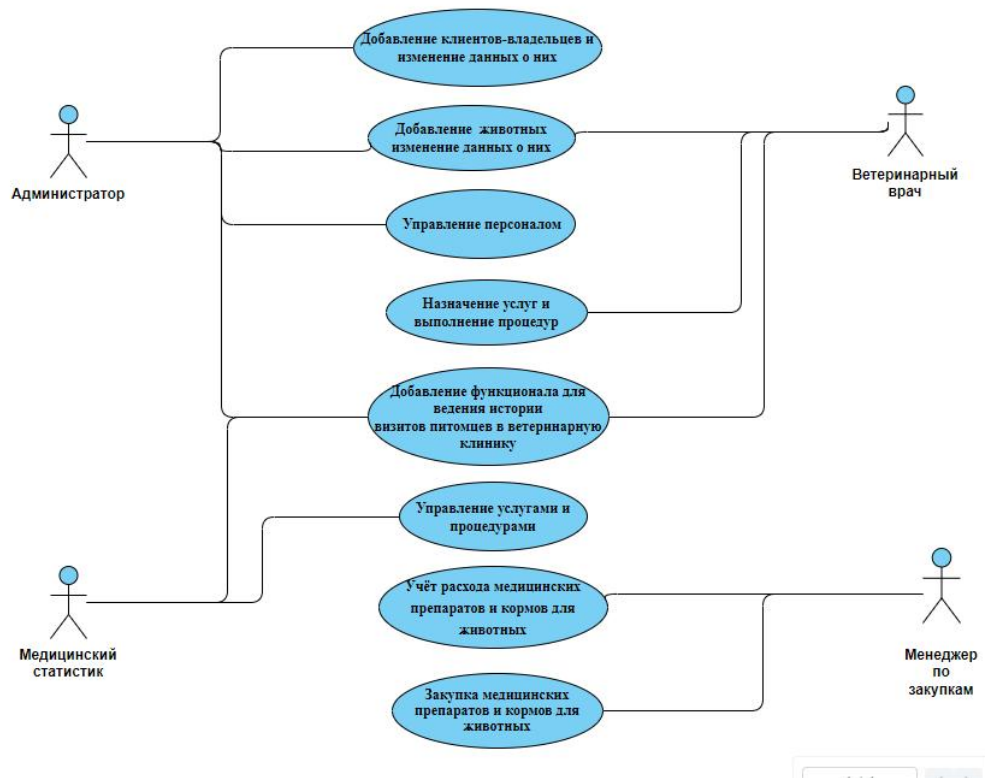


Рис.1.9 Диаграмма вариантов использования

С проектируемой системой будут взаимодействовать следующие действующие лица:

- администратор,
- менеджер по закупкам,
- медицинский статистик,
- ветеринарный врач.

Рассмотрим варианты использования каждого из действующих лиц.

Администратору доступны следующие функции:

- добавление животных и изменение данных о них;
- добавление клиентов и изменение данных о них;
- добавление новых посещений и изменение данных о них;
- учёт сотрудников ветеринарной клиники.

Взаиморасчёт с клиентом Медицинскому статистику доступны следующие функции:

- добавление функционала для ведения истории визитов питомцев в ветеринарную клинику;
- управление услугами и процедурами.

Менеджеру по закупкам доступны следующие функции:

1. учёт расхода медицинских препаратов и кормов для животных;
2. закупка медицинских препаратов и кормов для животных;

Ветеринарному врачу доступны следующие функции:

- добавление животных и изменение данных о них;
- просмотр клиентов и поиск по именам;
- просмотр услуг и их назначение;
- добавление новых посещений и их изменение.

В первом приближении для решения выделенных задач необходимо хранение данных о следующих объектах:

1. Животное – содержит данные о каждом животном, которому оказывалась услуга в ветеринарной клинике (имя, вид, порода, дата рождения, пол).
2. Ветеринарный врач – содержит личные данные о врачах в ветеринарной клинике (ФИО, дата рождения, телефон, должность/специальность, зарплата).
3. Клиент – содержит личные данные о каждом клиенте, который когда-либо записывался на приём (ФИО, адрес, телефонный номер, электронная почта, последнее посещение).
4. Посещение – содержит данные о последних посещениях владельцев питомцев в ветеринарную клинику (год, месяц, число, поставленный диагноз, оказанная услуга).
5. Услуга – содержит информацию об оказываемых услугах питомцам (название услуги, стоимость)
6. Медицинский препарат – содержит информацию о препаратах, закупаемые ветеринарной клиникой (название, производитель, количество, цена).

7. Корм – содержит информацию о кормах, закупаемые ветеринарной клиникой (название, производитель, тип корма, количество, цена).

8. Расход – содержит информацию о расходах медицинских препаратов и кормов для животных (название препарата, название корма, количество препаратов, количество корма).

9. Закупка – содержит информацию о закупке медицинских препаратов и кормов (дата закупки, поставщик, название корма, название препаратов, количество корма, количество препаратов, цена).

1.3. ЛОГИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

Логическое проектирование заключается в определении состава и структуры таблиц базы данных на основе результатов концептуального проектирования и проверки полученной модели с помощью методов нормализации.

ER-диаграмма (Entity-Relationship Diagram), или диаграмма "сущность-связь", является графическим инструментом для семантического моделирования предметной области. Она используется для визуализации структуры данных и отношений между различными сущностями в системе.

ER-диаграмма состоит из трех основных компонентов:

1. Сущности - объекты или концепции, которые могут быть идентифицированы в предметной области.
2. Атрибуты - характеристики или свойства, которые описывают сущности.
3. Связи - отношения между сущностями.

В ER-диаграммах выделяют следующие типы связей:

- Один-к-одному (1:1): Одна сущность связана с одной сущностью другого типа.
- Один-ко-многим (1:M): Одна сущность связана с несколькими сущностями другого типа.
- Многие-ко-многим (M:N): Несколько сущностей связаны с несколькими сущностями другого типа.

В ER-диаграммах сущности могут быть классифицированы по классам принадлежности:

- Обязательная сущность - сущность, которая должна иметь по крайней мере один экземпляр, связанный с экземпляром другой сущности через определенную связь.

- Необязательная сущность - сущность, которая может не иметь экземпляров, связанных с экземпляром другой сущности через определенную связь.

Благодаря концептуальному проектированию были выявлены основные объекты базы данных – сущности ER-проектирования. На основе результатов концептуального проектирования можно приступить к логическому проектированию и построению ER-диаграмм.

Построим ER-диаграммы всех сущностей и связей между ними.

Клиент должен иметь хотя бы одно животное, и у каждого животного обязательно должен иметься конкретный владелец (рис. 1.10).

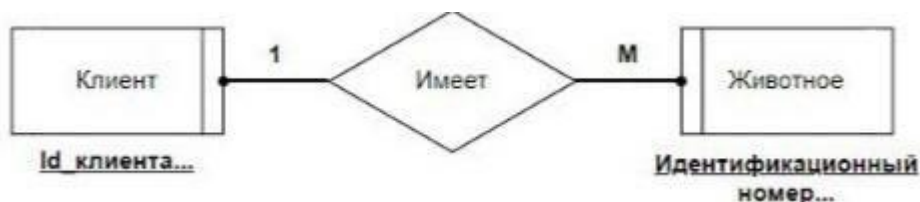


Рис.1.10 ER-диаграмма клиента и животного

Каждое животное приводят на посещение или не приводят (если не все посещения в ветеринарной клинике могут быть связаны с конкретным животным), и каждое посещение должно быть записано на конкретного животного (рис. 1.11).



Рис.1.11 ER-диаграмма животного и посещения

Ветеринарный врач осуществляет множество посещений, и каждое посещение должно осуществляться ветеринарным врачом (рис. 1.12).



Рис.1.12 ER-диаграмма животного и посещения

Каждое посещение предоставляет услуги, и каждая услуга может быть предоставлена на нескольких посещениях (рис. 1.13).



Рис.1.13 ER-диаграмма посещения и услуги

Посещение фиксирует расход препаратов и кормов или не фиксирует (если посещения могут быть консультацией и не требовать дополнительных расходов), и каждый расход должен быть связан с конкретным посещением (рис. 1.14)



Рис.1.14 ER-диаграмма посещения и расхода

Каждый расход может содержать определенный медицинский препарат или не содержать вовсе (если не все животные, обратившиеся в ветеринарную клинику, нуждаются в назначении определенного препарата или лечения), и медицинскому препарату может соответствовать несколько расходов или не соответствовать вовсе (рис. 1.15).



Рис.1.15 ER-диаграмма расходов и медицинского препарата

Каждый расход может содержать определенный корм или не содержать вовсе (если не все животные, обратившиеся в ветеринарную клинику, нуждаются в назначении определенного корма или лечения), и каждому корму может соответствовать несколько расходов или не соответствовать вовсе (рис. 1.16).



Рис.1.16 ER-диаграмма расходов и корма

Каждая закупка может содержать определенный медицинский препарат или не содержать вовсе (если клиника может получать препараты в виде пожертвований или производить их самостоятельно), и каждому медицинскому препарату может соответствовать несколько закупок или не соответствовать вовсе (рис. 1.17).



Рис.1.17 ER-диаграмма закупки и медицинского препарата

Каждая закупка может содержать определенный корм или не содержать вовсе (если клиника может получать корма в виде пожертвований или производить их самостоятельно), и каждому корму может соответствовать несколько закупок или не соответствовать вовсе (рис. 1.18).



Рис.1.18 ER-диаграмма закупки и корма

Деятельность, направленная на выявление реальных потребностей заказчика, а также на выяснения смысла высказанных требований, называется анализом предметной области (бизнес-моделированием, если речь идет о потребностях коммерческой организации). Анализ предметной области – это первый шаг этапа системного анализа, с которого начинается разработка программной системы.

Сформируем набор предварительных отношений с указанием предполагаемого первичного ключа для каждого отношения.

Связь *ИМЕЕТ* удовлетворяет условиям правила 4, в соответствии с которым получаем два отношения:

1. Клиент (Id_клиента, ...)
2. Животное (Идентифицированный номер, Id_клиента, ...) – добавился неключевой атрибут Id_клиента.

Связь *ПРИВОДЯТ НА* удовлетворяет условиям правила 4, в соответствии с которым получаем два отношения:

1. Животное (Идентифицированный номер, ...)
2. Посещение (Id_посещения, Идентифицированный номер ...) – добавился неключевой атрибут Идентифицированный номер.

Связь *ОСУЩЕСТВЛЯЕТ* удовлетворяет условиям правила 4, в соответствии с которым получаем два отношения:

1. Ветеринарный врач (Номер врача, ...)
2. Посещение (Id_посещения, Номер врача, ...) – добавился неключевой атрибут Номер врача.

Связь *ПРЕДОСТАВЛЯЕТ* удовлетворяет условиям правила 6, в соответствии с которым получаем два отношения:

1. Посещение (Id_посещения, ...)
2. Услуга (Номер услуги, ...)
3. ПосещениеУслуга (Id_посещения, Номер услуги)

Связь *ФИКСИРУЕТ* удовлетворяет условиям правила 2, в соответствии с которым получаем два отношения:

1. Посещение (Id_посещения, ...)
2. Расход (Id_расхода, Id_посещения, ...) – добавился неключевой атрибут Id_посещения.

Связь *СОДЕРЖИТ* удовлетворяет условиям правила 5, в соответствии с которым получаем два отношения:

1. Расход (Id_расхода, ...)
2. Медицинский препарат (Номер препарата, ...)
3. РасходПрепарат (Id_расхода, Номер препарата)

Связь *СОДЕРЖИТ2* удовлетворяет условиям правила 5, в соответствии с которым получаем два отношения:

1. Расход (Id_расхода, ...)
2. Корм (Номер корма, ...)
3. РасходКорм (Id_расхода, Номер корма)

Связь *СОДЕРЖИТ3* удовлетворяет условиям правила 5, в соответствии с которым получаем два отношения:

1. Закупка (Номер закупки, ...)
2. Медицинский препарат (Номер препарата, ...)
3. ЗакупкаПрепарат (Номер закупки, Номер препарата)

Связь *СОДЕРЖИТ4* удовлетворяет условиям правила 5, в соответствии с которым получаем два отношения:

1. Закупка (Номер закупки, ...)
2. Корм (Номер корма, ...)
3. ЗакупкаКорм (Номер закупки, Номер корма)

Добавим неключевые атрибуты в каждое из предварительных отношений с условием, чтобы отношения отвечали требованиям третьей нормальной формы.

Животное (Идентификационный номер, Id_клиента, имя, вид, порода, дата рождения, пол)

Ветеринарный врач (Номер врача, ФИО, дата рождения, телефон, должность/специальность, зарплата)

Клиент (Id_клиента, ФИО, адрес, телефонный номер, электронная почта)

Посещение (Id_посещения, Идентификационный номер, Номер врача, дата, диагноз, итоговая стоимость)

Услуга (Номер услуги, название услуги, стоимость)

Медицинский препарат (Номер препарата, название, производитель, количество, единица измерения, цена)

Корм (Номер корма, название, производитель, тип корма, количество, единица измерения, цена)

Расход (Id_расхода, Id_посещения)

Закупка (Номер закупки, дата закупки, поставщик, дата поставки, итого)

ПосещениеУслуга (Id_посещения, Номер услуги)

РасходПрепарат (Id_расхода, Номер препарата, количество)

РасходКорм (Id_расхода, Номер корма, количество)

ЗакупкаПрепарат (Номер закупки, Номер препарата, количество)

ЗакупкаКорм (Номер закупки, Номер корма, количество)

Ограничения предметной области, то есть известные как бизнес-правила, - это конкретные правила или условия, которые определяют или ограничивают различные аспекты бизнеса. Они обеспечивают структуру для управления, контроля и выполнения бизнес-процессов и операций.

Описание требований и ограничений ветеринарной клиники:

- Все ветеринарные врачи должны иметь должность.
- Все животные должны предоставить полную информацию, (имя, вид, порода, дата рождения, пол).
- Оказать услугу могут только те врачи, которые достигли возраста восемнадцати лет.
- Стоимость услуги должна быть больше 0, за исключением бесплатных осмотров.
- Итоговая стоимость посещения равна стоимости всех услуг посещения + стоимость израсходованных медицинских препаратов + стоимость израсходованного корма.
- Текущие или прошедшие даты должны стоять по умолчанию.
- Дата поставки закупки должна быть больше даты закупки.
- В посещение должна входить хотя бы одна услуга (например, осмотр).

- При подтверждении поступления закупки, количество корма или препарата на складе должно быть увеличено на количество, указанное в закупке.
- При фиксации расхода корма или препарата, количество на складе должно быть уменьшено на соответствующее количество, указанное в расходной записи.
- В расходе/закупке должен быть израсходован хотя бы один товар (корм/медикамент).
- При увольнении врача на его невыполненные задачи должен быть назначен новый.

1.4. ФИЗИЧЕСКОЕ ПРОЕКТИРОВАНИЕ

Физическое проектирование базы данных является одним из ключевых этапов процесса проектирования базы данных, который заключается в реализации уже существующей логической модели данных на конкретной СУБД. Основной целью физического проектирования является определение общей схемы БД (определение связей между таблицами), выбор типов данных и значений по умолчанию для каждого атрибута, определение ключей и иных ограничений.

В качестве СУБД для проектирования базы данных выбран MS SQL Server. Среда SQL Server Management Studio – это основной, стандартный и полнофункциональный инструмент для работы с Microsoft SQL Server, разработанный компанией Microsoft, который предназначен как для разработчиков, так и для администраторов SQL Server.

В частности, в SQL Server Management Studio можно определять атрибуты, их типы данных, ключи и другие ограничения. Также можно использовать SQL-запросы для создания таблиц и заполнения их данными.

На логическом этапе проектирования базы данных определяются отношения и их атрибуты. На физическом этапе проектирования логическая модель преобразуется в конкретную структуру таблиц, индексов, ключей и других элементов базы данных.

Таким образом, учитывая имеющиеся ограничения предметной области, получаем следующие таблицы (табл. 1.1 – 1.14).

Таблица 1.1

Требования к структуре таблицы «Клиент»

Столбец	Тип данных	Нуль?	Ключ	Ограничение
Id_клиента	Целое	Нет	Первичный	
ФИО	Строка	Нет		
Адрес	Строка	Нет		
Телефонный номер	Целое	Нет		Уникальный
Электронная почта	Строка	Нет		Уникальный

Таблица 1.2

Требования к структуре таблицы «Животное»

Столбец	Тип данных	Ноль?	Ключ	Ограничение	Ссылка
Идентификационный номер	Целое	Нет	Первичный Внешний		
Id_клиента	Целое	Нет			Id_клиента в Клиент
Имя	Строка	Нет			
Вид	Строка	Нет			
Порода	Строка	Нет			
Дата рождения	Дата	Нет		< текущей даты	
Пол	Строка	Нет		м, ж	

Таблица 1.3

Требования к структуре таблицы «Посещение»

Столбец	Тип данных	Ноль?	Ключ	Значение по умолч.	Ограничение	Ссылка
Id_посещения	Целое	Нет	Первичный			
Идентификационный номер	Целое	Нет	Внешний			Идентификационный номер в Животное
Номер врача	Целое	Нет	Внешний			Номер врача в Ветеринарный врач
Дата	Дата	Нет		Текущая дата	<= текущей даты	
Диагноз	Строка	Нет				
Итоговая стоимость	Целое	Нет			>=0	

Таблица 1.4

Требования к структуре таблицы «Услуга»

Столбец	Тип данных	Ноль?	Ключ	Значение по умолчанию	Ограничение
Номер услуги	Целое	Нет	Первичный		
Название услуги	Строка	Нет			Уникальный
Стоимость	Целое	Нет			>=0

Таблица 1.5

Требования к структуре таблицы «Ветеринарный врач»

Столбец	Тип данных	Нуль?	Ключ	Ограничение
Номер врача	Целое	Нет	Первичный	
ФИО	Строка	Нет		
Дата рождения	Дата	Нет		>=18 лет
Телефон	Целое	Нет		Уникальный
Должность_Специальность	Строка	Нет		
Зарплата	Целое	Нет		>30000

Таблица 1.6

Требования к структуре таблицы «Медицинский препарат»

Столбец	Тип данных	Нуль?	Ключ	Значение по умолчанию	Ограничение
Номер препарата	Целое	Нет	Первичный		
Название	Строка	Нет			
Производитель	Строка	Нет			
Количество	Целое	Нет			>0
Единица измерения	Строка	Нет			
Цена	Целое	Нет			>0

Таблица 1.7

Требования к структуре таблицы «Корм»

Столбец	Тип данных	Нуль?	Ключ	Ограничение
Номер корма	Целое	Нет	Первичный	
Название	Строка	Нет		
Производитель	Строка	Нет		
Тип корма	Строка	Нет		
Количество	Целое	Нет		>0
Единица измерения	Строка	Нет		
Цена	Целое	Нет		>0

Таблица 1.8

Требования к структуре таблицы «Расход»

Столбец	Тип данных	Нуль?	Ключ	Ограничение	Ссылка
Id_расхода	Целое	Нет	Первичный Внешний		
Id_посещения	Целое	Нет		Уникальный	Id_посещения в Посещение

Таблица 1.9

Требования к структуре таблицы «Закупка»

Столбец	Тип данных	Нуль?	Ключ	Значение по умолчанию	Ограничение
Номер закупки	Целое	Нет	Первичный		
Дата закупки	Дата	Нет			
Поставщик	Строка	Нет			
Дата поставки	Дата	Нет			Дата_поставки > Дата_закупки
Итого	Целое	Нет			

Таблица 1.10

Требования к структуре таблицы «ПосещениеУслуга»

Столбец	Тип данных	Нуль?	Ключ	Ссылка
Id_посещения	Целое	Нет	Первичный Внешний	Id_посещения в Посещение
Номер услуги	Целое	Нет		Номер услуги в Услуга

Таблица 1.11

Требования к структуре таблицы «РасходПрепарат»

Столбец	Тип данных	Нуль?	Ключ	Ограничение	Ссылка
Id_расхода	Целое	Нет	Первичный		Id_расхода в Расход
Номер препарата	Целое	Нет	Внешний		Номер препарата в Медицинский препарат
Количество	Целое	Нет		>0	

Таблица 1.12

Требования к структуре таблицы «РасходКорм»

Столбец	Тип данных	Ноль?	Ключ	Ограничение	Ссылка
Id_расхода	Целое	Нет	Первичный		Id_расхода в Расход
Номер корма	Целое	Нет	Внешний		Номер корма в Корм
Количество	Целое	Нет		>0	

Таблица 1.13

Требования к структуре таблицы «ЗакупкаПрепарат»

Столбец	Тип данных	Ноль?	Ключ	Ограничение	Ссылка
Номер закупки	Целое	Нет	Первичный		Номер закупки в Закупка
Номер препарата	Целое	Нет	Внешний		Номер препарата в Медицинский препарат
Количество	Целое	Нет		>0	

Таблица 1.14

Требования к структуре таблицы «ЗакупкаКорм»

Столбец	Тип данных	Ноль?	Ключ	Ограничение	Ссылка
Номер закупки	Целое	Нет	Первичный		Номер закупки в Закупка
Номер корма	Целое	Нет	Первичный		Номер корма в Корм
Количество	Целое	Нет		>0	

Заполнение таблиц данными представлено в приложении А..

Общая схема БД, показывающая связи таблиц и их состав представлена на рис. 1.19.

У таблицы Посещение есть две хранимые процедуры dbo.AddVisit и [dbo].РасчитатьИтоговуюСтоимость (Приложение Б, п.3, п.19). Процедура dbo.AddVisit необходима для добавления в БД нового посещения. Входными данными являются id посещения, идентификационный номер, номер врача, дата, диагноз и итоговая стоимость. Внутри процедуры данные проверяются на корректность. Процедура dbo.РасчитатьИтоговуюСтоимость необходима для расчёта итоговой стоимости посещения, учитывая стоимость услуги, которая была оказана, купленный корм и медицинский препарат. Входными данными является id посещения.

У таблицы Ветеринарный врач есть одна хранимая процедура dbo.AddVetDoctor (Приложение Б, п.4). Процедура dbo.AddVetDoctor необходима для добавления в БД нового ветеринарного врача. Входными данными являются номер врача, ФИО, дата рождения, телефон, должность/специальность и зарплата. Внутри процедуры данные проверяются на корректность.

У таблицы Услуга есть один триггер dbo.trgCheckServiceCost (Приложение Б, п.5). Триггер dbo.trgCheckServiceCost необходим для добавления в БД новой услуги. Входными данными являются номер услуги, название услуги и стоимость. Внутри процедуры данные проверяются на корректность.

У таблицы Корм есть три хранимых процедуры dbo.AddFood, dbo.Рассчитать_Общее_Количество_Корма и dbo.Рассчитать_Сумму_Корма (Приложение Б, п.6, п.15, п.16). Процедура dbo.AddFood необходима для добавления в БД нового корма. Входными данными являются номер корма, название, производитель, тип корма, количество, единица измерения, цена. Внутри процедуры данные проверяются на корректность. Процедура dbo.Рассчитать_Общее_Количество_Корма необходима для изменения количества корма на складе после закупки или расхода. Входными данными является номер корма. Процедура dbo.Рассчитать_Сумму_Корма необходима

для изменения цены корма от общего количества после закупки или расхода. Входными данными является номер корма.

У таблицы Медицинский препарат есть три хранимые процедуры `dbo.AddMedicalDrug`, `dbo.Рассчитать_Общее_Количество_Препарата` и `dbo.Рассчитать_Сумму_Препарата` (Приложение Б, п.7, п.17, п.18). Процедура `dbo.AddMedicalDrug` необходима для добавления в БД нового медицинского препарата. Входными данными являются номер препарата, название, производитель, количество, единица измерения, цена. Внутри процедуры данные проверяются на корректность. Процедура `dbo.Рассчитать_Общее_Количество_Препарата` необходима для изменения нового количества препарата на складе после закупки или расхода. Входными данными является номер препарата. Процедура `dbo.Рассчитать_Сумму_Препарата` необходима для изменения цены препарата от общего количества после закупки или расхода. Входными данными является номер препарата.

У таблицы Закупка есть две хранимые процедуры `dbo.AddZakupka` и `dbo.Рассчитать_Итоговую_Сумму_Закупки` (Приложение Б, п.8, п.14). Процедура `dbo.AddZakupka` необходима для добавления в БД новой закупки. Внутри процедуры данные проверяются на корректность. Входными данными являются номер закупки, дата закупки, поставщик, дата поставки и итог. Процедура `dbo.Рассчитать_Итоговую_Сумму_Закупки` необходима для расчёта суммы после закупки корма или препарата. Входными данными является номер закупки.

У таблицы ЗакупкаКорм есть одна хранимая процедура `dbo.AddZakupkaFood` (Приложение Б, п.9). Процедура `dbo.AddZakupkaFood` необходима для добавления в БД новой закупки корма. Входными данными являются номер закупки, номер корма и количество. Внутри процедуры данные проверяются на корректность.

У таблицы ЗакупкаПрепарат есть одна хранимая процедура `dbo.AddZakupkaDrug` (Приложение Б, п.10). Процедура `dbo.AddZakupkaDrug`

необходима для добавления в БД новой закупки препарата. Входными данными являются номер закупки, номер препарата и количество. Внутри процедуры данные проверяются на корректность.

У таблицы Расход есть одна хранимая процедура dbo.AddRashod (Приложение Б, п.11). Процедура dbo.AddRashod необходима для добавления в БД нового расхода. Входными данными являются id расхода и id посещения. Внутри процедуры данные проверяются на корректность.

У таблицы РасходКорм есть одна хранимая процедура dbo.AddRashodKorm (Приложение Б, п.12). Процедура dbo.AddRashodKorm необходима для добавления в БД нового расхода корма. Входными данными являются id расхода, номер корма и количество. Внутри процедуры данные проверяются на корректность.

У таблицы РасходПрепарат есть одна хранимая процедура dbo.AddRashodPreparat (Приложение Б, п.13). Процедура dbo.AddRashodPreparat необходима для добавления в БД нового расхода препарата. Входными данными являются id расхода, номер препарата и количество. Внутри процедуры данные проверяются на корректность.

2. ОПИСАНИЕ ФУНКЦИОНИРОВАНИЯ БД

2.1. НАЗНАЧЕНИЕ И ПЕРЕЧЕНЬ ФУНКЦИЙ БАЗЫ ДАННЫХ

Главной и важнейшей задачей при разработке программной реализации интерфейса работы с базой данных является, в первую очередь, создание интуитивно понятного, простого и удобного интерфейса для пользователя для его взаимодействия с базой данных. Данный интерфейс создан для четырёх пользователей – администратора, ветеринарного врача, менеджера по закупкам и медицинского статистика. С помощью интерфейса они могут взаимодействовать с базой данных.

Администратор имеет функционал, такой как:

- добавление животных и изменение данных о них;
- добавление клиентов и изменение данных о них;
- добавление новых посещений и изменение данных о них;
- учёт сотрудников ветеринарной клиники.
- взаиморасчёт с клиентом.

Ветеринарный врач имеет следующие возможности:

- добавление животных и изменение данных о них;
- просмотр клиентов и поиск по именам;
- просмотр услуг и их назначение;
- добавление новых посещений и их изменение.

Медицинский статистик имеет следующие возможности:

- добавление новых услуг и изменение данных о них;
- просмотр услуг;
- назначение услуг;
- добавление новых посещений и изменение данных о них.

Менеджер по закупкам имеет следующий функционал:

- учёт корма;
- учёт медицинских препаратов;
- учёт закупки корма и медицинских препаратов;

- учёт расхода корма и медицинских препаратов;
- просмотр посещений в ветеринарную клинику;
- расчёт количества и цены корма на складе после закупки или расхода;
- расчёт количества и цены медицинского препарата на складе после закупки или расхода.

Для реализации пользовательского интерфейса использовались: среда разработки Visual Studio 2019 [6] и язык программирования C# [4].

2.2. ОПИСАНИЕ РАБОТЫ С БАЗОЙ ДАННЫХ

При входе в приложение открывается главная страница (рис. 2.1).

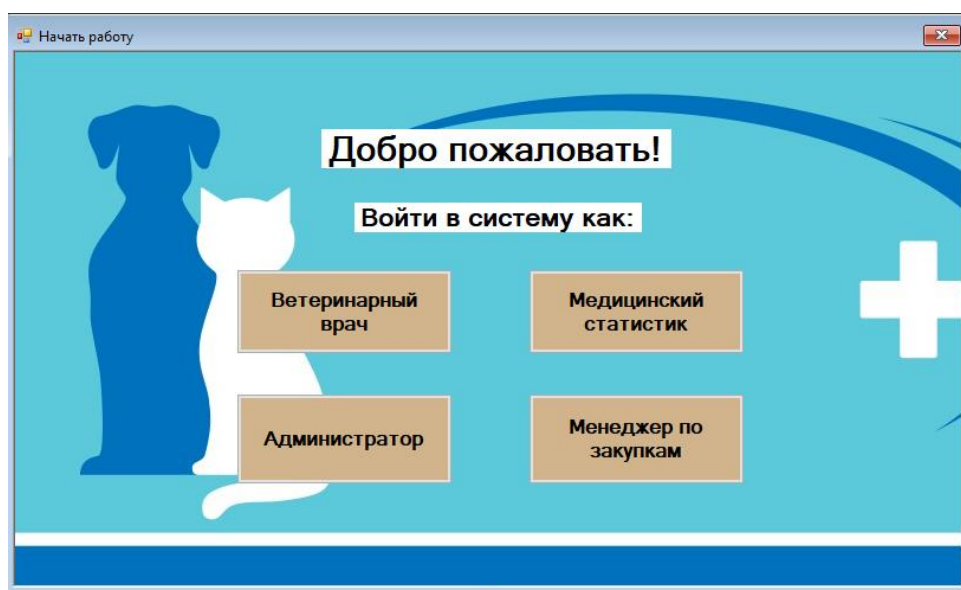


Рис. 2.1 Главное окно

Пользователю доступны четыре функции начала работы. При нажатии кнопки Администратор, открывается форма Администратора и появляется окно с клиентами (рис. 2.2).

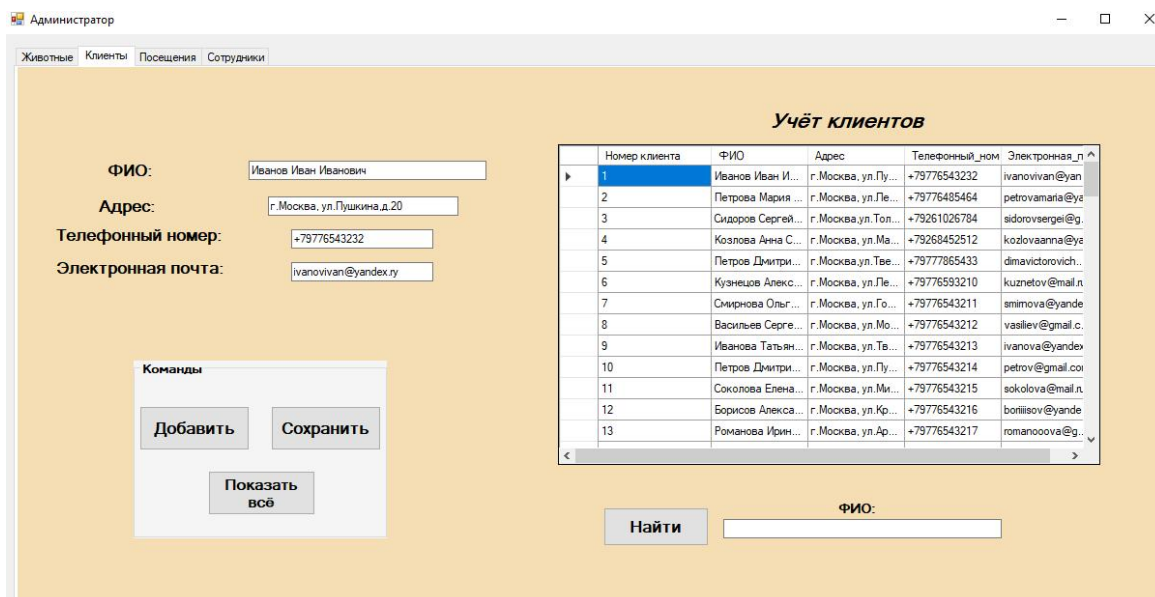


Рис. 2.2 Окно с клиентами

На этом этапе пользователю доступны действия, такие как:

1. Добавление клиента;
2. Сохранение изменений;
3. Просмотр клиентов;

4. Поиск клиента по ФИО.

Начнем с добавления клиента. При нажатии на кнопку срабатывает хранимая процедура `dbo.AddClient` (приложение Б). Заполнив поля на открывшейся форме, пользователь выполнит нужное действие (рис. 2.3-2.4).

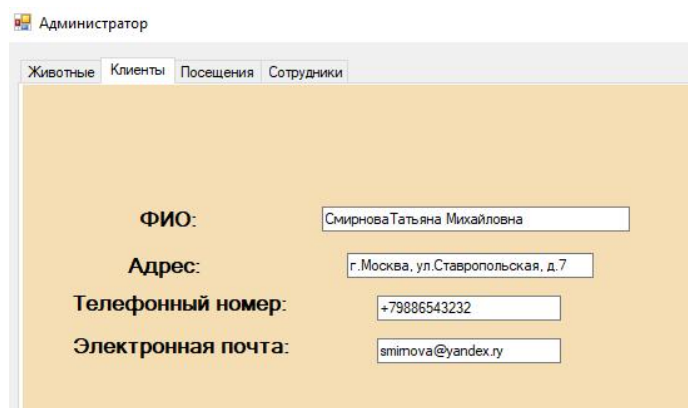


Рис. 2.3 Добавление нового клиента

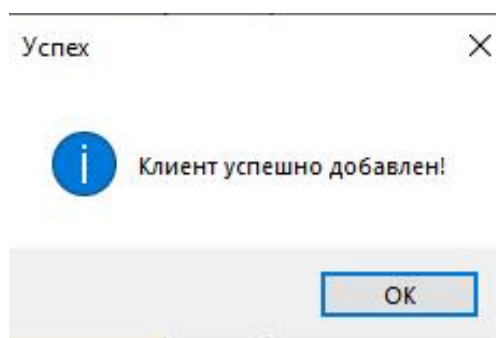


Рис. 2.4 Сообщение об успешном добавлении

Если заполнить поля некорректно, то хранимая процедура выдаст ошибку (рис. 2.5-2.7).

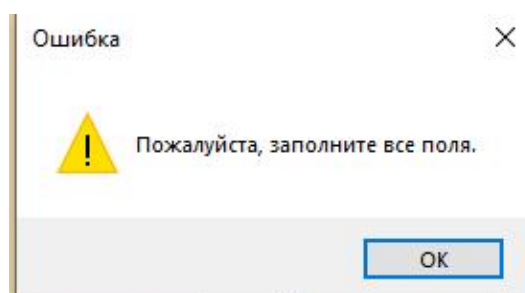


Рис. 2.5 Сообщение об ошибке

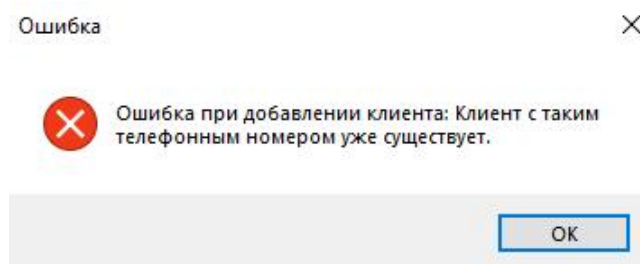


Рис. 2.6 Ошибка при повторяющимся номере

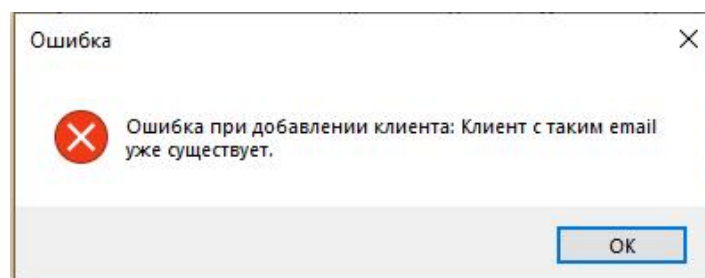


Рис. 2.7 Ошибка при повторяющимся email

Для поиска создано отдельное поле в нижней части формы и кнопка «Найти». Поиск осуществляется по всем возможным совпадениям ФИО клиента (рис. 2.8).

Учёт клиентов

	Номер клиента	ФИО	Адрес	Телефонный_ном	Электронная_поч
▶	7	Смирнова Ольг...	г.Москва, ул.Го...	+79776543211	smimova@yande...
	26	Смирнова Ирин...	г.Москва, ул.Ма...	+79776543230	smimoova@mail.ru
	64	Смирнова Тать...	г.Москва, ул.Ма...	+79776543268	smmimova@mail.ru
	104	СмирноваТатья...	г.Москва, ул.Ст...	+79886543232	smimova@yande...

Найти

ФИО:

Рис. 2.8 Поиск клиентов по ФИО

Кнопка «Показать всё» позволяет увидеть всех клиентов таблицы (рис. 2.9).

Учёт клиентов

ФИО:
Адрес:
Телефонный номер:
Электронная почта:

Команды

Номер клиента	ФИО	Адрес	Телефонный_ном	Электронная_п
1	Иванов Иван И...	г.Москва, ул.Пу...	+79776543232	ivanovivan@yan
2	Петрова Мария ...	г.Москва, ул.Ле...	+79776485464	petrovamaria@yc
3	Сидоров Сергей...	г.Москва, ул.Тол...	+79261026784	sidorovsergei@g
4	Козлова Анна С...	г.Москва, ул.Ма...	+79268452512	kozlovaanna@ya
5	Петров Дмитри...	г.Москва, ул.Тее...	+79777865433	dimavictorovich..
6	Кузнецов Алекс...	г.Москва, ул.Ле...	+79776593210	kuznetov@mail.n
7	Смирнова Ольг...	г.Москва, ул.Го...	+79776543211	smimova@yande
8	Васильев Серге...	г.Москва, ул.Мо...	+79776543212	vasiliev@gmail.c
9	Иванова Татьян...	г.Москва, ул.Та...	+79776543213	ivanova@yandex
10	Петров Дмитри...	г.Москва, ул.Пу...	+79776543214	petrov@gmail.coi
11	Соколова Елена...	г.Москва, ул.Ми...	+79776543215	sokolova@mail.n
12	Борисов Алекса...	г.Москва, ул.Кр...	+79776543216	borisov@yande
13	Романова Ирин...	г.Москва, ул.Ар...	+79776543217	romanoova@g...

Рис. 2.9 Кнопка «Показать всё»

Кнопка «Сохранить» позволяет сохранить изменения в данных клиента (рис. 2.10-2.11).

ФИО:
Адрес:
Телефонный номер:
Электронная почта:

Команды

Рис. 2.10 Изменение номера телефона клиента

▶	104	СмирноваТатья...	г.Москва, ул.Ст...	+79886543200	smimova@yande
---	-----	------------------	--------------------	--------------	---------------

Рис. 2.11 Сохранённые изменения

Далее пользователь может перейти на вкладку Животные (рис. 2.12).

Учёт животных

Номер животного	Id_клиента	ФИО	Имя	Вид	Порода	Дата_рождения	Пол
1	1	Иванов Иван И...	Борман	Кот	Шотландский ви...	01.03.2009	м
2	2	Петрова Мария ...	Белка	Собака	Йоркширский т...	20.05.2021	ж
3	3	Сидоров Сергей...	Рыжик	Кот	Сиамская	10.08.2022	м
4	4	Козлова Анна С...	Жучка	Собака	Немецкая овч...	15.10.2021	ж
5	5	Петров Дмитри...	Муся	Кошка	Домус	04.01.2019	ж
6	6	Кузнецов Алекс...	Барсик	Кот	Сибирский	15.07.2018	м
7	7	Смирнова Ольг...	Лайма	Собака	Мопс	02.11.2020	ж
8	8	Васильев Серге...	Пушок	Кот	Персидский	28.03.2021	м
9	9	Иванова Татьян...	Тоша	Собака	Джек-рассел-те...	10.09.2019	м
10	10	Петров Дмитри...	Зоя	Кошка	Шотландская ви...	17.01.2022	ж
11	11	Соколова Елена...	Рекс	Собака	Лабрадор-ретри...	05.05.2020	м
12	12	Борисов Алекса...	Мася	Кошка	Британская	24.12.2019	ж
13	13	Романова Иван	Шляк	Собака	Сибирская халки	14.02.2021	м

Рис. 2.12 Окно с животными

На этом этапе пользователю доступны действия, такие как:

1. Добавление животного;
2. Сохранение изменений;
3. Просмотр животных;
4. Поиск животного по ФИО клиента-хозяина.

Начнем с добавления животного. При нажатии на кнопку срабатывает хранимая процедура `dbo.AddAnimal` (приложение Б). Заполнив поля на открывшейся форме, пользователь выполнит нужное действие (рис. 2.13-2.14).

Добавление нового животного

Номер клиента: 104

Имя: Миша

Вид: Кот

Порода: Домус

Дата рождения: 10 августа 2022 г.

Пол: м

Рис. 2.13 Добавление нового животного

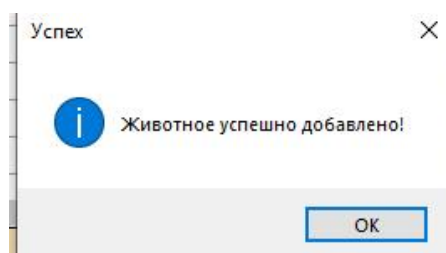


Рис. 2.14 Сообщение об успешном добавлении

Если заполнить поля некорректно, то хранящая процедура выдаст ошибку (рис. 2.15-2.16).

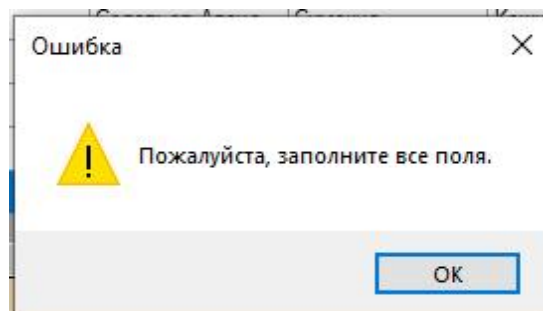


Рис. 2.15 Сообщение об ошибке

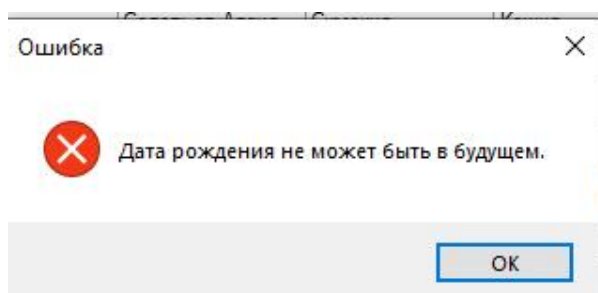


Рис. 2.16 Ошибка при некорректной дате рождения

Для поиска создано отдельное поле в нижней части формы и кнопка «Найти». Поиск осуществляется по всем возможным совпадениям ФИО клиента-хозяина (рис. 2.17).

Учёт животных

	Номер животного	Id_клиента	ФИО	Имя	Вид	Порода	Дата_рождения	Пол
▶	7	7	Смирнова Ольг...	Лайма	Собака	Мопс	02.11.2020	ж
	26	26	Смирнова Ирин...	Барон	Собака	Немецкая овч...	10.11.2019	м
	64	64	Смирнова Тать...	Пушистик	Кот	Персидский	15.07.2020	м
	104	104	Смирнова Татя...	Миша	Кот	Домус	10.08.2022	м

ФИО хозяина:

Смирнова

Найти

Рис. 2.17 Поиск клиента-хозяина по ФИО

Кнопка «Показать всё» позволяет увидеть всех животных таблицы (рис. 2.18).

Выход

Номер клиента:

Имя:

Вид:

Порода:

Дата рождения:

Пол:

Команды

Учёт животных

	Номер животного	Id_клиента	ФИО	Имя	Вид	Порода	Дата_рождения	Пол
▶	1	1	Иванов Иван И...	Борман	Кот	Шотландский ви...	01.03.2009	м
	2	2	Петрова Мария...	Белка	Собака	Йоркширский т...	20.05.2021	ж
	3	3	Сидоров Сергей...	Рыжик	Кот	Сиамская	10.08.2022	м
	4	4	Козлова Анна С...	Жучка	Собака	Немецкая овч...	15.10.2021	ж
	5	5	Петров Дмитри...	Муся	Кошка	Домус	04.01.2019	ж
	6	6	Кузнецов Алекс...	Барсик	Кот	Сибирский	15.07.2018	м
	7	7	Смирнова Ольг...	Лайма	Собака	Мопс	02.11.2020	ж
	8	8	Васильев Серге...	Пушок	Кот	Персидский	28.03.2021	м
	9	9	Иванова Татьяна...	Тоша	Собака	Джек-рассел-те...	10.09.2019	м
	10	10	Петров Дмитри...	Зоя	Кошка	Шотландская ви...	17.01.2022	ж
	11	11	Соколова Елена...	Рекс	Собака	Паблдор-ретри...	05.05.2020	м
	12	12	Борисов Алекса...	Мася	Кошка	Британская	24.12.2019	ж
	13	13	Романова Илья...	Шалки	Собака	Сибирская хаски	14.02.2021	м

ФИО хозяина:

Найти

Рис. 2.18 Кнопка «Показать всё»

Кнопка «Сохранить» позволяет сохранить изменения в данных животных (рис. 2.19-2.20).

Номер клиента: 103

Имя: Миша

Вид: Кот

Порода: Домус

Дата рождения: 10 августа 2022 г.

Пол: М

Рис. 2.19 Изменение номера клиента

104	103	Иванов Николай Иванович	Миша	Кот	Домус	10.08.2022	М
-----	-----	-------------------------	------	-----	-------	------------	---

Рис. 2.20 Сохранённые изменения

Далее пользователь может перейти на вкладку Посещения (рис. 2.21).

Администратор

Животные Клиенты **Посещения** Сотрудники

Номер животного: 1

Номер врача: 1

Дата: 12 марта 2023 г.

Диагноз: Аллергия

Итоговая стоимость: 5200

Команды: Добавить Сохранить Показать все

Учёт Посещения

Id посещения	Номер животного	Имя животного	Номер врача	ФИО врача	Дата	Диагноз
1	1	Борман	1	Козлов Алексан...	12.03.2023	Аллергия
2	2	Белка	2	Смирнова Юлия...	13.03.2023	Растяжение
3	3	Рыжик	3	Смирнов Денис...	14.03.2023	Остеохондроз
4	4	Жучка	4	Андреева Елена...	17.03.2023	Аллергия
5	5	Мурка	5	Егоров Андрей...	25.03.2023	Гематома
6	6	Барсик	6	Иванов Иван И...	26.03.2023	Простуда
7	7	Лайка	7	Петрова Мария...	27.03.2023	Перелом г
8	8	Пушок	8	Сидоров Сергей...	28.03.2023	Конъюнктив
9	9	Тоша	9	Козлова Анна С...	29.03.2023	Дерматит
10	10	Зоя	11	Кузнецова Олга...	30.03.2023	Глисты

Найти Id посещения: 1

Расчёт Id посещения: 1 Итоговая сумма:

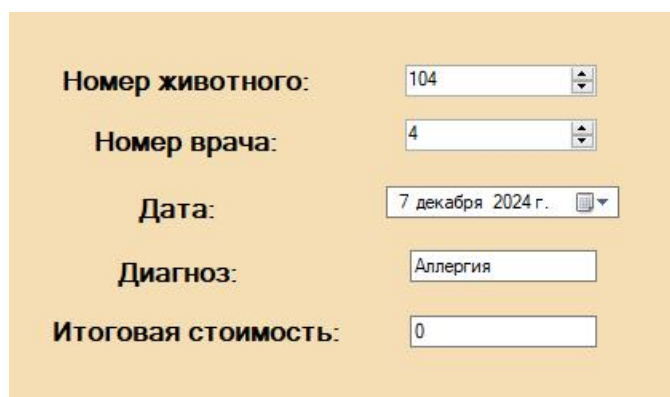
Рис. 2.21 Окно с посещениями

На этом этапе пользователю доступны действия, такие как:

1. Добавление посещения;
2. Сохранение изменений;
3. Просмотр всех посещений;
4. Поиск конкретного посещения по его номеру.
5. Расчёт итоговой стоимости посещения.

Начнем с добавления посещения. При нажатии на кнопку срабатывает хранимая процедура `dbo.AddVisit` (приложение Б). Заполнив поля на

открывшейся форме, пользователь выполнит нужное действие (рис. 2.22-2.23).



The form is titled 'Добавление нового посещения' (Adding a new visit). It contains the following fields:

- Номер животного: 104
- Номер врача: 4
- Дата: 7 декабря 2024 г.
- Диагноз: Аллергия
- Итоговая стоимость: 0

Рис. 2.22 Добавление нового посещения

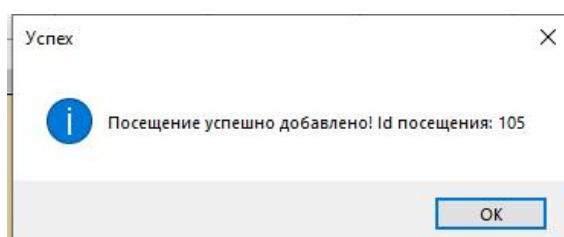


Рис. 2.23 Сообщение об успешном добавлении

Если заполнить поля некорректно, то хранящая процедура выдаст ошибку (рис. 2.24-2.25).

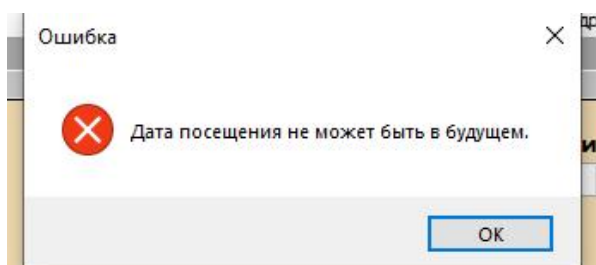


Рис. 2.24 Ошибка при некорректной дате посещения

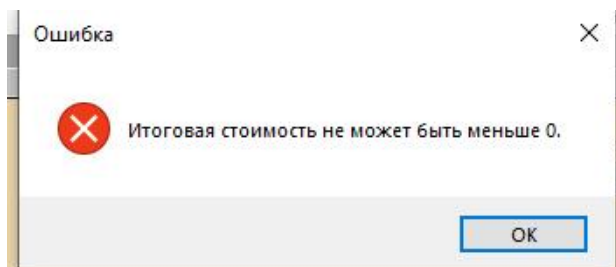


Рис. 2.25 Ошибка при некорректной итоговой стоимости

Для поиска создано отдельное поле в нижней части формы и кнопка «Найти». Поиск осуществляется по номеру посещения (рис. 2.26).

Учёт Посещения

	Id_посещения	Номер животного	Имя животного	Номер_врача	ФИО врача	Дата	Диагноз
▶	105	104	Миша	4	Андреева Елиза...	07.12.2024	Аллергия

Найти

Id посещения:

Рис. 2.26 Поиск посещения по номеру

Кнопка «Показать всё» позволяет увидеть все посещения таблицы (рис. 2.27).

Учёт Посещения

Номер животного:

Номер врача:

Дата:

Диагноз:

Итоговая стоимость:

Команды

Добавить
Сохранить

Показать
всё

	Id_посещения	Номер животного	Имя животного	Номер_врача	ФИО врача	Дата	Диагноз
▶	1	1	Борман	1	Козлов Алексан...	12.03.2023	Аллергия
	2	2	Белка	2	Смирнова Юлия...	13.03.2023	Растяжён
	3	3	Рыжик	3	Смирнов Денис...	14.03.2023	Остеохонд
	4	4	Жучка	4	Андреева Елиза...	17.03.2023	Аллергия
	5	5	Муся	5	Егоров Андрей ...	25.03.2023	Гематома
	6	6	Барсик	6	Иванов Иван И...	26.03.2023	Простуда
	7	7	Лайма	7	Петрова Мария ...	27.03.2023	Перелом л
	8	8	Пушок	8	Сидоров Сергей...	28.03.2023	Конъюнкт
	9	9	Тоша	9	Козлова Анна С...	29.03.2023	Дерматит
	10	10	Зоя	11	Кузнецова Олыг...	30.03.2023	Глисты

Найти

Id посещения:

Расчёт

Id посещения:
 Итоговая сумма:

Рис. 2.27 Кнопка «Показать всё»

Кнопка «Сохранить» позволяет сохранить изменения в данных посещений (рис. 2.28-2.29).

Номер животного:

Номер врача:

Дата:

Диагноз:

Итоговая стоимость:

Рис. 2.28 Изменение диагноза

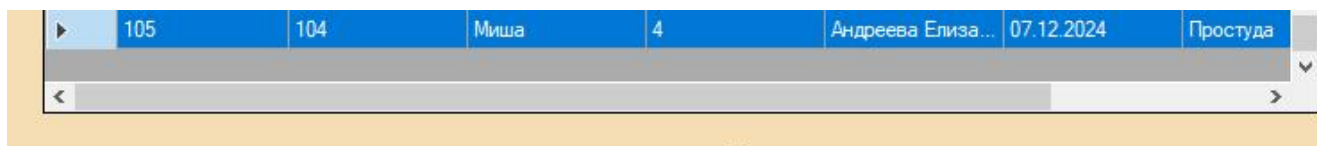


Рис. 2.29 Сохранённые изменения

Далее пользователь может перейти на вкладку Сотрудники (рис. 2.30).

Учёт Сотрудников

ФИО: Козлов Александр Сергеевич

Дата рождения: 1 января 1985 г.

Телефон: +79123457865

Должность/Специальность: Хирург

Зарплата: 80000

Команды: Добавить, Сохранить, Удалить, Показать все

Номер_врача	ФИО	Дата_рождения	Телефон	Должность_Спец	Зарплата
1	Козлов Алексан...	01.01.1985	+79123457865	Хирург	80000
2	Смирнова Юлия...	07.07.1990	+79261348969	Терапевт	65000
3	Смирнов Денис...	29.10.1989	+79758463793	Ортопед	75000
4	Андреева Елиза...	06.11.1980	+79269264829	Дерматолог	70000
5	Егоров Андрей...	11.12.1975	+79128482943	Хирург	80000
6	Иванов Иван И...	10.05.1980	+79261234567	Терапевт	70000
7	Петрова Мария...	25.08.1985	+79775556677	Хирург	85000
8	Сидоров Сергей...	18.02.1992	+79127890123	Ортопед	75000
9	Козлова Анна С...	04.11.1988	+79264567890	Дерматолог	65000
10	Петров Дмитри...	12.03.1978	+79121234567	Хирург	60000
11	Кузнецова Ольг...	01.06.1990	+79771112233	Терапевт	60000

Найти: ФИО: [Поиск]

Рис. 2.30 Окно с сотрудниками

На этом этапе пользователю доступны действия, такие как:

1. Добавление ветеринарного врача;
2. Сохранение изменений;
3. Просмотр сотрудников;
4. Поиск врача по его номеру.
5. Удаление ветеринарного врача.

Начнем с добавления сотрудника. При нажатии на кнопку срабатывает хранимая процедура `dbo.AddVetDoctor` (приложение Б). Заполнив поля на открывшейся форме и нажав на кнопку, пользователь выполнит нужное действие (рис. 2.31-2.32).

ФИО: Соловьёв Иван Семёнович

Дата рождения: 1 января 1985 г.

Телефон: +79123457000

Должность/Специальность: Терапевт

Зарплата: 80000

Рис. 2.31 Добавление нового сотрудника

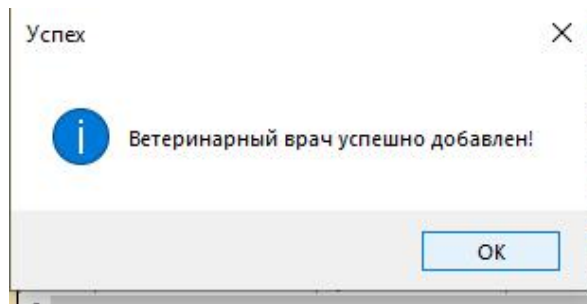


Рис. 2.32 Сообщение об успешном добавлении

Если заполнить поля некорректно, то хранящая процедура выдаст ошибку (рис. 2.33-2.34).

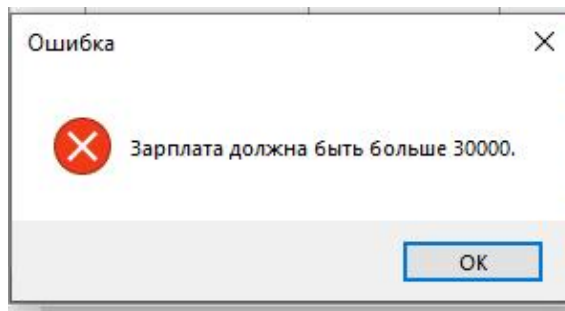


Рис. 2.33 Ошибка при некорректной зарплате

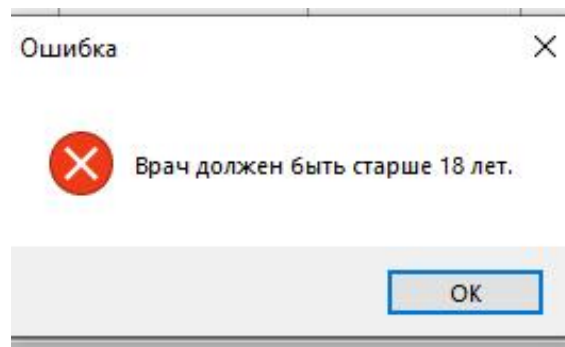


Рис. 2.34 Ошибка при некорректной дате рождения

Для поиска создано отдельное поле в нижней части формы и кнопка «Найти». Поиск осуществляется по ФИО врача (рис. 2.35).

Учёт Сотрудников

	Номер_врача	ФИО	Дата_рождения	Телефон	Должность_Спец	Зарплата
▶	12	Соловьёв Иван ...	01.01.2007	+79123457000	Терапевт	80000

<
>

Найти

ФИО:

Рис. 2.35 Поиск ветеринарного врача по ФИО

Кнопка «Показать всё» позволяет увидеть всех сотрудников таблицы (рис. 2.36).

ФИО:

Дата рождения:

Телефон:

Должность/Специальность:

Зарплата:

Команды

Добавить
Сохранить

Удалить
Показать всё

Учёт Сотрудников

	Номер_врача	ФИО	Дата_рождения	Телефон	Должность_Спец	Зарплата
▶	1	Козлов Алексан...	01.01.1985	+79123457865	Хирург	80000
	2	Смирнова Юлия...	07.07.1990	+79261348969	Терапевт	65000
	3	Смирнов Денис...	29.10.1989	+79758463793	Ортопед	75000
	4	Андреева Елиза...	06.11.1980	+79269264829	Дерматолог	70000
	5	Егоров Андрей ...	11.12.1975	+79128482943	Хирург	80000
	6	Иванов Иван И...	10.05.1980	+79261234567	Терапевт	70000
	7	Петрова Мария ...	25.08.1985	+79775556677	Хирург	85000
	8	Сидоров Сергей...	18.02.1992	+79127890123	Ортопед	75000
	9	Козлова Анна С...	04.11.1988	+79264567890	Дерматолог	65000
	10	Петров Дмитри...	12.03.1978	+79121234567	Хирург	60000
	11	Кузнецова Ольг...	01.06.1990	+79771112233	Терапевт	60000

<
>

Найти

ФИО:

Рис. 2.36 Кнопка «Показать всё»

Кнопка «Сохранить» позволяет сохранить изменения в данных сотрудников (рис. 2.37-2.38).

ФИО:

Дата рождения:

Телефон:

Должность/Специальность:

Зарплата:

Рис. 2.37 Изменение зарплаты

11	Полученная услуга ...	01.01.1985	+79123457000	Терапевт	60000
12	Соловьёв Иван ...	01.01.1985	+79123457000	Терапевт	60000

Рис. 2.38 Сохранённые изменения

С помощью кнопки «Выход» пользователь может вернуться в главное окно. На этом функции администратора заканчиваются, поэтому переходим к функциям ветеринарного врача.

На этом этапе ветеринару доступны следующие действия:

1. Добавление и изменение данных животных;
2. Добавление и изменение данных посещений;
3. Просмотр списка сотрудников ветеринарной клиники;
4. Просмотр списка клиентов ветеринарной клиники;
5. Поиск клиента по ФИО;
6. Просмотр списка услуг;
7. Назначение услуг каждому посещению;

Первые пять действий совпадают с функциями администратора, поэтому перейдём сразу к услугам. В окне «Услуги» ветеринарному врачу доступен просмотр списка существующих услуг (рис. 2.39).

Ветеринарный врач

Животные Услуги Посещения

Услуги

Номер_услуги	Название_услуги	Стоимость
1	Хирургия	1510
2	Вакцинация	1500
3	Терапия	700
4	Анализы	500
5	Бесплатный ос...	0
6	Стерилизация	2500
7	Кастрация	2000
8	УЗИ	1000
9	Рентген	1000
10	Чистка зубов	1200

Рис. 2.39 Просмотр услуг

Заполнив поля на открывшейся форме и нажав на кнопку «Добавить», пользователь выполнит нужное действие. Подтверждение добавления осуществляется кнопкой «Сохранить» (рис. 2.40-2.41).

Id посещения: 105
Номер услуги: 2
Название услуги: Вакцинация

Добавить Найти

Сохранить

Id посещения: 1

Рис. 2.40 Добавление услуги в посещение

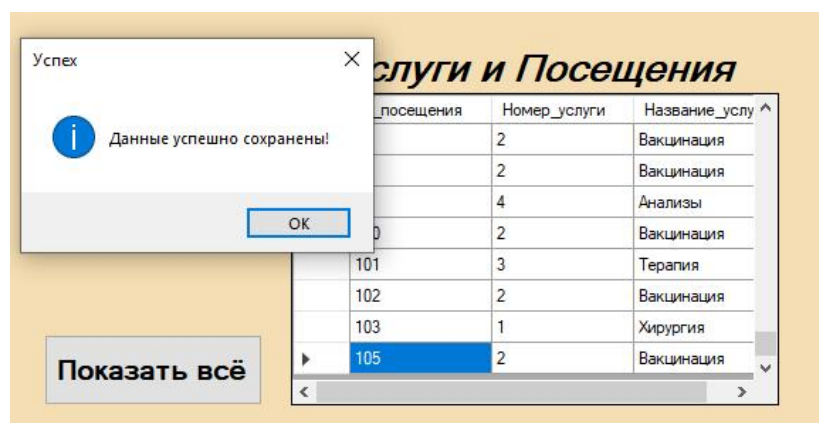


Рис. 2.41 Сообщение об успешном добавлении

При неверном вводе данных, система выводит сообщение об ошибке (рис.2.42-2.43).

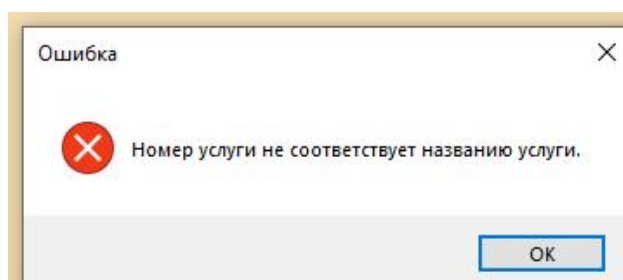


Рис. 2.42 Ошибка при некорректном вводе номера услуги

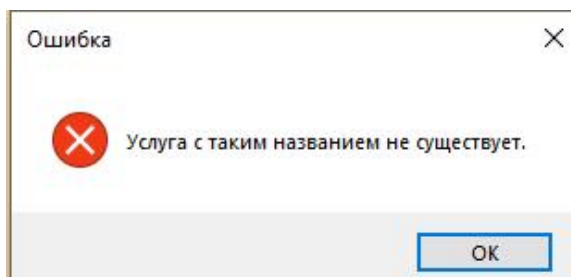


Рис. 2.43 Ошибка при некорректном вводе названия услуги

Для поиска создано отдельное поле в верхней части формы и кнопка «Найти». Поиск осуществляется по номеру посещения (рис. 2.44).

	Id посещения	Номер услуги	Название услуги
▶	105	2	Вакцинация

Всё

Рис. 2.44 Поиск по номеру посещения

Кнопка «Показать всё» позволяет увидеть все посещения и оказанные услуги в таблице (рис. 2.45).

	Id посещения	Номер услуги	Название услу ^
▶	1	1	Хирургия
	1	3	Терапия
	1	4	Анализы
	1	5	Бесплатный ос.
	2	3	Терапия
	2	5	Бесплатный ос.
	3	3	Терапия
	4	2	Вакцинация

Показать всё

Рис. 2.45 Кнопка «Показать всё»

На этом функции ветеринарного врача заканчиваются. Перейдем к функциям медицинского статистика. Ему доступны следующие действия:

1. Добавление и изменение данных посещений;
2. Просмотр списка сотрудников ветеринарной клиники;

3. Просмотр списка животных;
4. Назначение услуг каждому посещению;
5. Добавление и изменение данных услуг;
6. Поиск животного по имени;

Первые четыре действия совпадают с функциями ветеринарного врача, поэтому перейдём к услугам в окне «Услуги».

Начнем с добавления новой услуги. При нажатии на кнопку «Добавить» срабатывает триггер `dbo.trgCheckServiceCost` (приложение Б). Заполнив поля на открывшейся форме, пользователь выполнит нужное действие (рис. 2.46-2.47).

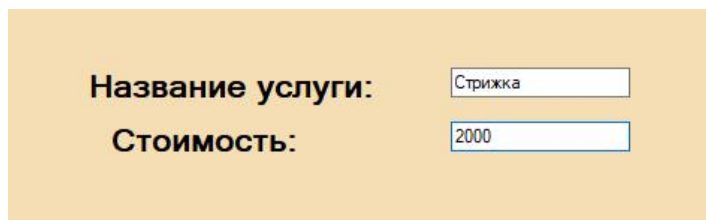


Рис. 2.46 Добавление новой услуги

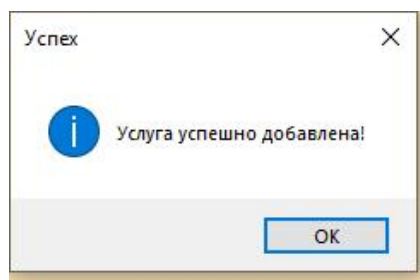


Рис. 2.47 Сообщение об успешном добавлении

При неверном вводе данных, система выводит сообщение об ошибке (рис.2.48-2.49).

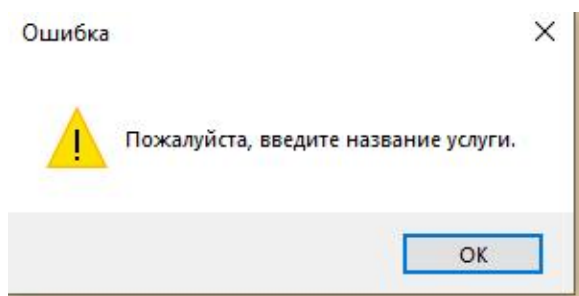


Рис. 2.48 Ошибка при некорректном вводе данных

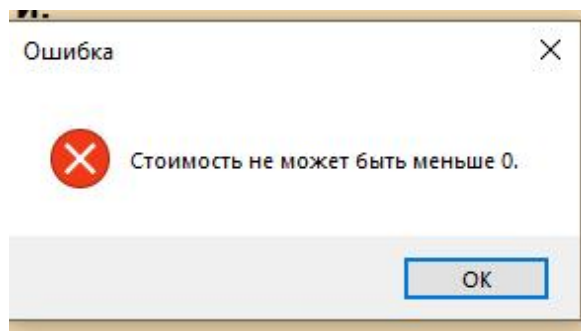


Рис. 2.49 Ошибка при некорректном вводе стоимости услуги

Кнопка «Сохранить» позволяет сохранить изменения в данных услуг (рис. 2.50-2.51).

Название услуги:

Стоимость:

Рис. 2.50 Изменение стоимости услуги

11	Стрижка	2500
----	---------	------

Рис. 2.51 Сохранённые изменения

В окне «Посещения» пользователь может найти нужное животное по его имени с помощью кнопки «Найти» (рис. 2.52).

Животные

	Номер животного	Id_клиента	ФИО	Имя	Вид	Порода
▶	104	104	Смирнова Татьяна...	Миша	Кот	Домус

Имя:

Найти

Рис. 2.52 Поиск животного по имени

Кнопка «Показать всё» возвращает таблицу в исходное состояние (рис. 2.53).

Животные

Имя:

Показать всё **Найти**

Номер животного	Id_клиента	ФИО	Имя	Вид	Порода
1	1	Иванов Иван И...	Борман	Кот	Шотландский в
2	2	Петрова Мария ...	Белка	Собака	Йоркширский т
3	3	Сидоров Сергей...	Рыжик	Кот	Сиамская
4	4	Козлова Анна С...	Жучка	Собака	Немецкая овч
5	5	Петров Дмитри...	Муся	Кошка	Домус
6	6	Кузнецов Алекс...	Барсик	Кот	Сибирский
7	7	Смирнова Олг...	Лайма	Собака	Мопс
8	8	Васильев Серге...	Паша	Кот	Персидский

Рис. 2.53 Кнопка «Показать всё»

На этом функции медицинского статистика завершены. Перейдем к возможностям менеджера по закупкам. Пользователю на данном этапе доступны следующие функции:

1. Учёт корма;
2. Учёт медицинских препаратов;
3. Учёт закупаемого корма;
4. Учёт закупаемого препарата;
5. Добавление и изменение закупки;
6. Поиск закупки по дате закупки;
7. Учёт израсходованного корма;
8. Учёт израсходованного медицинского препарата;
9. Просмотр посещений;
10. Расчёт количества и цены корма на складе после закупки или расхода;
11. Расчёт количества и цены медицинского препарата на складе после закупки или расхода.

Начнём с добавления корма. При нажатии на кнопку срабатывает хранимая процедура `dbo.AddFood` (приложение Б). Заполнив поля на открывшейся форме, пользователь выполнит нужное действие (рис. 2.54-2.55).

Название:	<input type="text" value="Test"/>
Производитель:	<input type="text" value="Test"/>
Тип корма:	<input type="text" value="Сухой"/>
Количество:	<input type="text" value="10"/>
Единица измерения:	<input type="text" value="кг"/>
Цена:	<input type="text" value="35000"/>

Рис. 2.54 Добавление нового корма

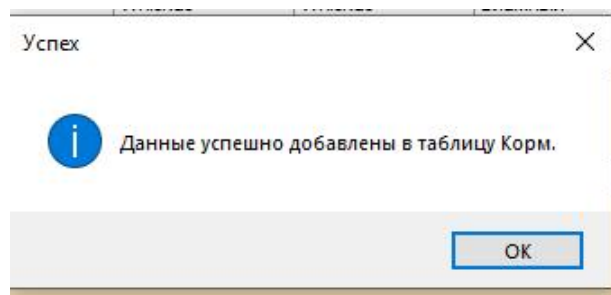


Рис. 2.55 Сообщение об успешном добавлении корма

При неверном вводе данных, система выводит сообщение об ошибке (рис.2.56-2.57).

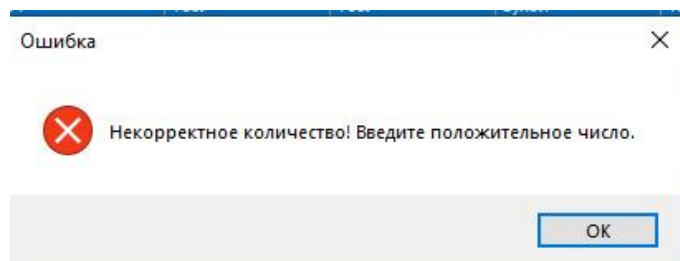


Рис. 2.56 Ошибка при вводе некорректного количества корма

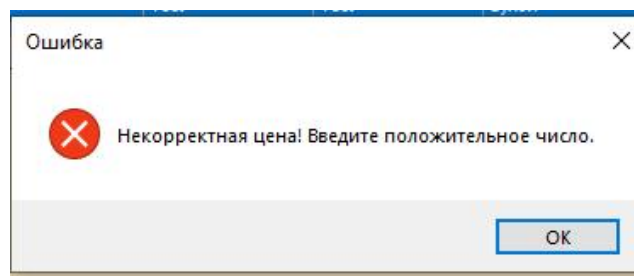


Рис. 2.57 Ошибка при вводе некорректной цены корма

Пользователь может выбрать нужный корм по его названию из выпадающего списка с помощью кнопки «Найти» (рис. 2.58).

Учёт Корма

	Номер_корма	Название	Производитель	Тип_корма	Количество	Единица_измерен	Цена
▶	11	Test	Test	Сухой	10	кг	35000

Название

Рис. 2.58 Поиск корма по его названию

Перейдём в другое окно «Закупка корма» и добавим новую закупку. При нажатии на кнопку «Добавить» срабатывает хранимая процедура `dbo.AddZakupka` (приложение Б). Первоначально наша итоговая стоимость закупки будет равна нулю и обновится только после закупки корма или медицинского препарата. Заполнив поля на открывшейся форме, пользователь выполнит нужное действие (рис. 2.59-2.62).

Дата закупки:

Поставщик:

Дата поставки:

Итого:

Рис. 2.59 Добавление новой закупки

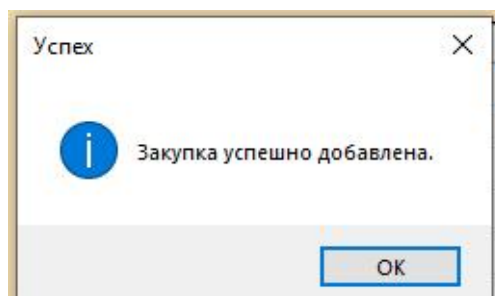


Рис. 2.60 Сообщение об успешном добавлении закупки

	102	09.12.2024	Test	10.12.2024	0
--	-----	------------	------	------------	---

Рис. 2.61 Добавленная закупка

При неверном вводе данных, система выводит сообщение об ошибке (рис.2.62).

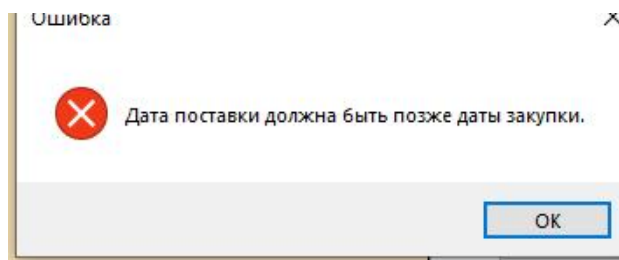


Рис. 2.62 Ошибка при некорректном вводе даты закупки и поставки

Перейдём к самой закупке корма. При нажатии на кнопку «Добавить» срабатывает хранимая процедура `dbo.AddZakupkaFood` (приложение Б). Заполнив поля на открывшейся форме, пользователь выполнит нужное действие (рис. 2.63-2.64).

Рис. 2.63 Добавление закупки корма

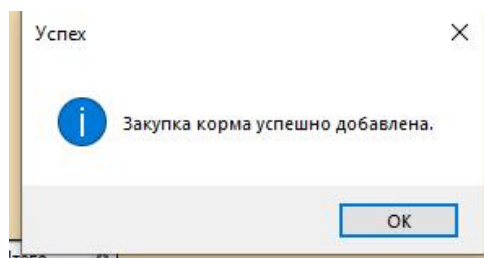


Рис. 2.64 Сообщение об успешном добавлении закупки корма

При неверном вводе данных, система выводит сообщение об ошибке (рис.2.65-2.66).

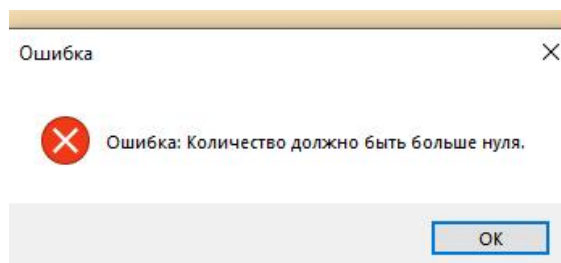


Рис. 2.65 Ошибка при некорректном вводе количества корма

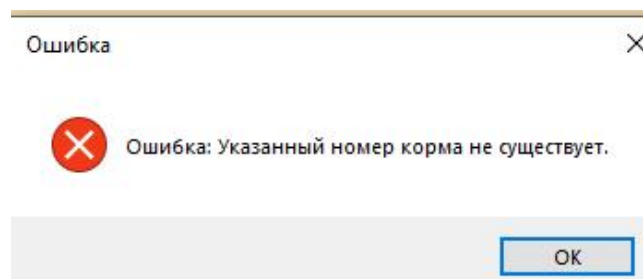


Рис. 2.66 Ошибка при некорректном вводе номера корма

Аналогичные действия сделаем с медицинскими препаратами. При нажатии на кнопку «Добавить» срабатывает хранимая процедура `dbo.AddMedicalDrug` (приложение Б). Заполнив поля на открывшейся форме и нажав на кнопку, пользователь выполнит нужное действие (рис. 2.67-2.68).

Рис. 2.67 Добавление нового медицинского препарата

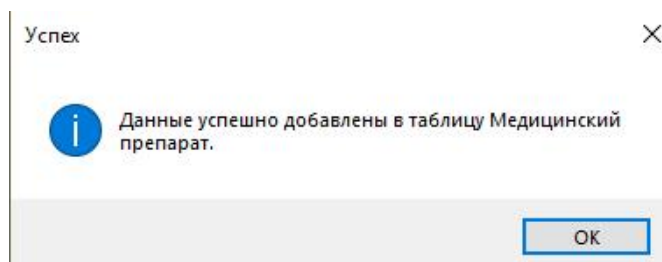


Рис. 2.68 Сообщение об успешном добавлении медицинского препарата

При неверном вводе данных, система выводит сообщение об ошибке (рис.2.69-2.70).

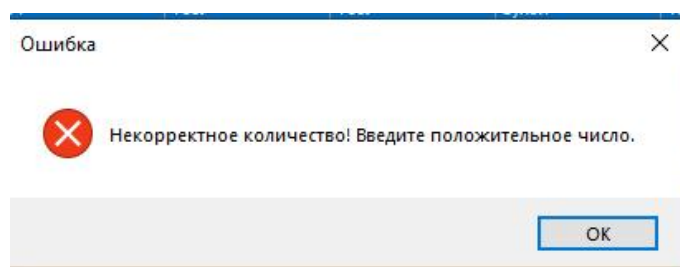


Рис. 2.69 Ошибка при вводе некорректного количества препарата

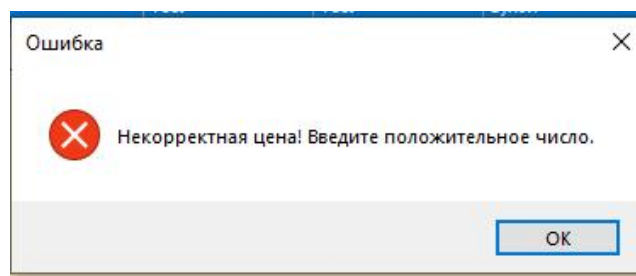


Рис. 2.70 Ошибка при вводе некорректной цены препарата

Пользователь может выбрать нужный медицинский препарат по его названию из выпадающего списка с помощью кнопки «Найти» (рис. 2.71).

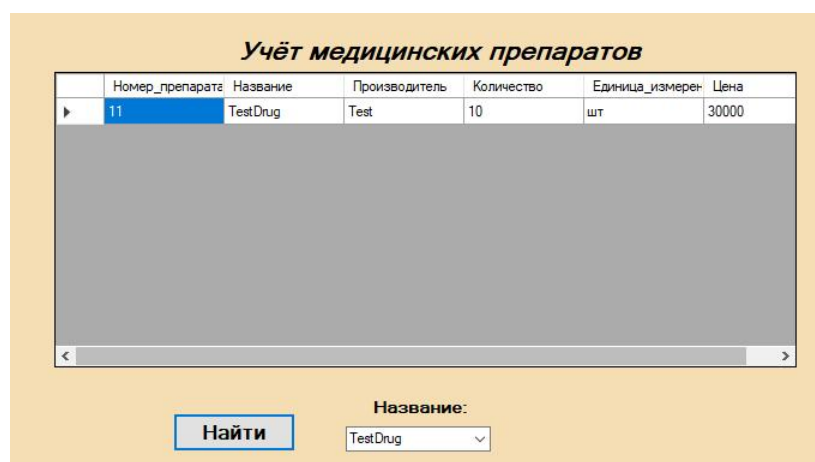


Рис. 2.71 Поиск препарата по названию

Перейдём к закупке препарата. При нажатии на кнопку «Добавить» срабатывает хранимая процедура `dbo.AddZakupkaDrug` (приложение Б). Заполнив поля на открывшейся форме и нажав на кнопку, пользователь выполнит нужное действие (рис. 2.72-2.73).

Рис. 2.72 Добавление закупки препарата

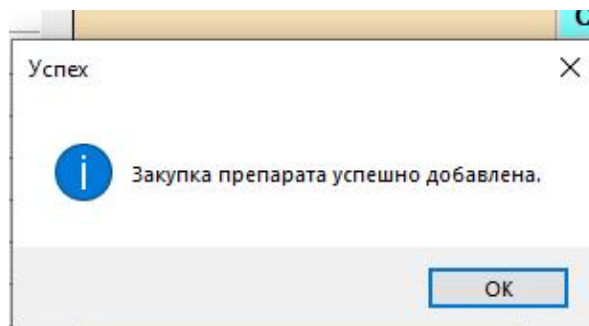


Рис. 2.73 Сообщение об успешном добавлении закупки препарата

При неверном вводе данных, система выводит сообщение об ошибке (рис.2.74-2.75).

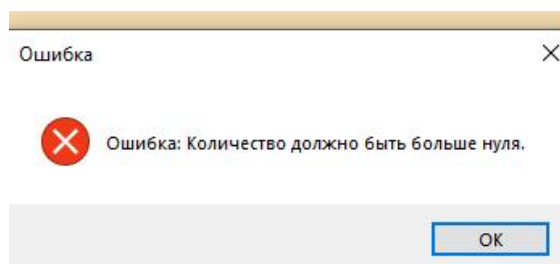


Рис. 2.74 Ошибка при некорректном вводе количества препарата

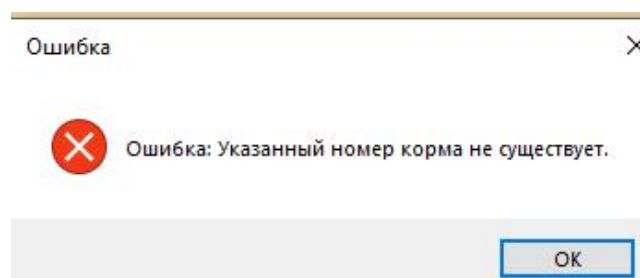


Рис. 2.75 Ошибка при некорректном вводе номера препарата

Теперь, когда закупка корма и препарата осуществлена, пользователь может рассчитать итоговую сумму закупки. При нажатии на кнопку «Рассчитать сумму закупки» срабатывает хранимая процедура [dbo].Рассчитать_Итоговую_Сумму_Закупки (приложение Б). И введя нужный номер, выведется новая сумма (рис.2.76).

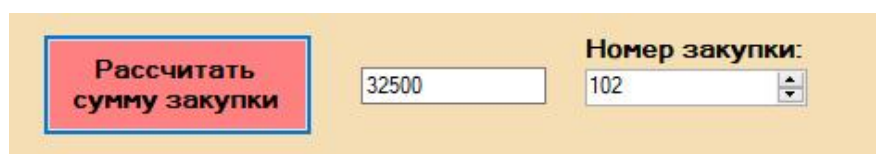


Рис. 2.76 Расчёт итоговой суммы закупки

Вернёмся к корму. Поскольку осуществилась закупка корма, то количество и цена на складе должны измениться.

При нажатии на кнопку «Рассчитать количество и цену» срабатывают хранимые процедуры [dbo].Рассчитать_Общее_Количество_Корма и [dbo].Рассчитать_Сумму_Корма (приложение Б). Указав нужный номер корма, работает кнопка (рис. 2.77).

Рис. 2.77 Расчёт нового количества и цены корма

Аналогично для медицинского препарата. При нажатии на кнопку «Рассчитать количество и цену» срабатывают хранимые процедуры [dbo].Рассчитать_Общее_Количество_Препарата и [dbo].Рассчитать_Сумму_Препарата (приложение Б). Указав нужный номер препарата, работает кнопка (рис. 2.78).

Рис. 2.78 Расчёт нового количества и цены препарата

Перейдём в другое окно «Расход корма» и добавим новый расход. При нажатии на кнопку «Добавить» срабатывает хранимая процедура dbo.AddRashod (приложение Б). Заполнив поля на открывшейся форме и нажав на кнопку, пользователь выполнит нужное действие (рис. 2.79-2.80).

Рис. 2.79 Добавление нового расхода

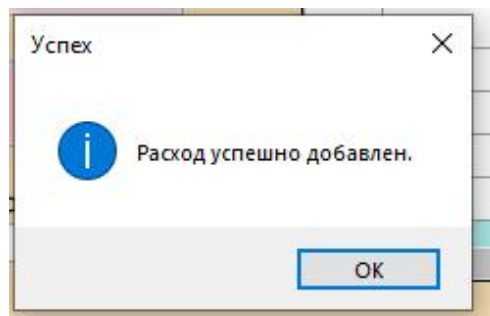


Рис. 2.80 Сообщение об успешном добавлении расхода

Перейдём к расходу корма. При нажатии на кнопку «Добавить» срабатывает хранимая процедура `dbo.AddRashodKorm` (приложение Б). Заполнив поля на открывшейся форме и нажав на кнопку, пользователь выполнит нужное действие (рис. 2.81-2.82).

Рис. 2.81 Добавление расхода корма

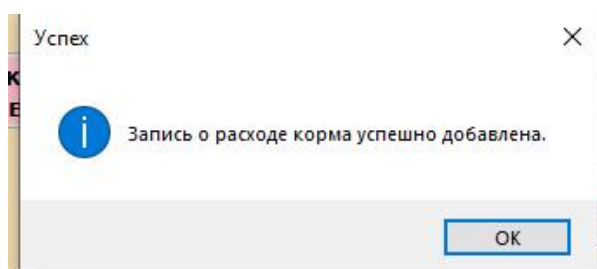


Рис. 2.82 Сообщение об успешном добавлении расхода корма

Аналогичные действия выполним для медицинского перпарата. При нажатии на кнопку «Добавить» срабатывает хранимая процедура `dbo.AddRashodPreparat` (приложение Б). Заполнив поля на открывшейся форме и нажав на кнопку, пользователь выполнит нужное действие (рис. 2.83-2.84).

Рис. 2.83 Добавление расхода препарата

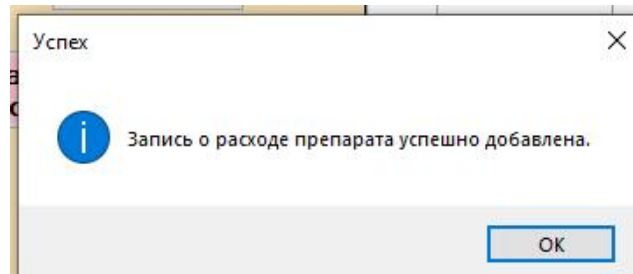


Рис. 2.84 Сообщение об успешном добавлении расхода препарата

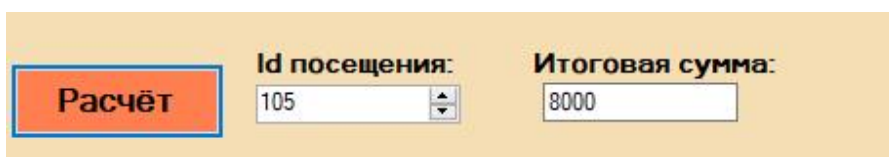
Ещё раз убедимся, что кнопка «Рассчитать количество и цену» в окне «Корм» и «Медицинский препарат» работает корректно (рис. 2.85-2.86).

Рис. 2.85 Пересчёт количества и цены корма

Рис. 2.86 Пересчёт количества и цены препарата

Вернёмся к функциям администратора. В окне «Посещения» с помощью кнопки Расчёт выведется итоговая стоимость посещения, не забывая указать конкретный номер посещения. Нажав на кнопку работает хранимая процедура [dbo].РассчитатьИтоговуюСтоимость (приложение Б).

Она учитывает стоимость услуг, которые были оказаны; количество купленного корма и препарата (рис. 2.87).



The image shows a user interface for calculating the total cost of a visit. It features a light orange background. On the left, there is a red button with a blue border labeled "Расчёт". To its right, there are two labels: "Id посещения:" and "Итоговая сумма:". Below "Id посещения:" is a text input field containing the number "105" and a small dropdown arrow icon. Below "Итоговая сумма:" is a text input field containing the number "8000".

Рис. 2.87 Расчёт итоговой стоимости посещения

ЗАКЛЮЧЕНИЕ

Основной тематикой данной работы является создание базы данных ветеринарной клиники.

Создание базы данных для ветеринарной клиники является актуальной темой в современном мире, поскольку она способствует повышению эффективности работы, улучшению обслуживания клиентов и их животных и принятия обоснованных решений. База данных позволяет ветеринарной клинике централизованно управлять всеми аспектами бизнеса, от отслеживания посещения до учёта взаимодействия ветеринарных врачей и клиентов.

В ходе выполнения проекта были получены следующие результаты:

1. Анализ предметной области позволил собрать информацию о уже имеющихся решениях и на их основе выделить те функции, которые необходимо реализовать. Для анализа рассматривались следующие ПО: конфигурация «Ветеринарная клиника» от «Простой Софт», а также сайт «VetMaster». Каждый из рассматриваемых продуктов имел свои особенности и возможности.

2. На этапе концептуального проектирования затронуты такие аспекты, как описание модели предметной области в общем виде. В процессе проектирования были выделены такие действующие лица, как администратор, менеджер по закупкам, медицинский статистик и ветеринарный врач, а также перечень функций, которые доступны каждому лицу. На основе анализа ДВИ был определен общий список объектов, о которых необходимо хранить информацию.

3. На основе уже имеющихся сущностей, их взаимосвязей и существующих правил преобразования мы создаем полноценный список отношений, в котором имеются как ключевые, так и неключевые атрибуты. Взаимосвязи между объектами описываются с использованием ER-диаграмм. В результате выполнения данного этапа проектирования было построено восемнадцать различных ER-диаграмм, которые описывают схему

взаимодействия между объектами. Отдельной задачей являлось определение ограничений предметной области. В результате мы получили свод правил, используемых в конкретной предметной области.

4. На этапе физического проектирования, используя среду SQL Server Management Studio, были созданы таблицы, настроены их связи взаимодействия друг с другом, а также было выполнено наполнение созданных таблиц данными. Таким образом, была получена полноценная база данных.

5. На заключительном этапе был разработан пользовательский интерфейс для взаимодействия с базой данных с помощью среды разработки Visual Studio 2019 и языка программирования C#. На основе данных, которые были определены в процессе проектирования, была разработана БД, позволяющая выполнить различные функции в зависимости от роли пользователя. Для Администратора доступны следующие функции: добавление животных и изменение данных о них, добавление клиентов и изменение данных о них, добавление новых посещений и изменение данных о них, учёт сотрудников ветеринарной клиники и взаиморасчёт с клиентом. Ветеринарный врач имеет некоторые функции администратора, дополнительно ему доступно просмотр клиентов и поиск по именам, просмотр услуг и их назначение. Медицинский статистик может добавлять новые услуги, изменять их и назначать. Для менеджера по закупкам доступны следующие функции: учёт корма, учёт медицинских препаратов, учёт закупки корма и медицинских препаратов, учёт расхода корма и медицинских препаратов, расчёт количества и цены корма на складе после закупки или расхода и расчёт количества и цены медицинского препарата на складе после закупки или расхода

В итоге был получен программный продукт, полностью соответствующий поставленным задачам.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Простой софт [Электронный ресурс] – 2024. – Режим доступа: <https://prostoysoft.ru/Veterinary.htm>, свободный. Дата обращения: 11.12.2024 г.
2. Ветеринарная клиника [Электронный ресурс] – 2024. – Режим доступа: <https://vetmaster.ru/>, свободный. Дата обращения: 11.12.2024 г.
3. Документация по c# [Электронный ресурс] – Microsoft, 2024. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/>. Дата обращения: 11.12.2024 г.
4. Программирование на c# [Электронный ресурс] – 2019. – Режим доступа: <https://metanit.com/>, свободный. Дата обращения: 11.12.2024 г.
5. Документация по SQL [Электронный ресурс] – Microsoft, 2022. – Режим доступа: <https://learn.microsoft.com/ru-ru/sql/?view=sql-serverver16>. Дата обращения: 11.12.2024 г.
6. Создание приложения Windows Forms на C# в Visual Studio [Электронный ресурс] – 2023. – Режим доступа: <https://learn.microsoft.com/ru-ru/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022>, свободный. Дата обращения: 11.12.2024 г.

ЗАПОЛНЕНИЕ ТАБЛИЦ ДАННЫМИ

	Id_клиента	ФИО	Адрес	Телефонный...	Электронная...
1	1	Иванов Иван ...	г.Москва, ул.П...	+79776543232	ivanovivan@ya...
2	2	Петрова Мари...	г.Москва, ул.Л...	+79776485464	petrovamaria@...
3	3	Сидоров Серге...	г.Москва, ул.То...	+79261026784	sidorovsergei@...
4	4	Козлова Анна ...	г.Москва, ул.М...	+79268452512	kozlovaanna@y...
5	5	Петров Дмитр...	г.Москва, ул.Тв...	+79777865433	dimavictorovic...
6	6	Кузнецов Алек...	г.Москва, ул.Л...	+79776593210	kuznetov@mail...
7	7	Смирнова Оль...	г.Москва, ул.Г...	+79776543211	smirnova@yan...
8	8	Васильев Серг...	г.Москва, ул.М...	+79776543212	vasiliev@gmail...
9	9	Иванова Татя...	г.Москва, ул.Т...	+79776543213	ivanova@yand...
10	10	Петров Дмитр...	г.Москва, ул.П...	+79776543214	petrov@gmail...
11	11	Соколова Елен...	г.Москва, ул.М...	+79776543215	sokolova@mail...
12	12	Борисов Алек...	г.Москва, ул.К...	+79776543216	borisov@yan...
13	13	Романова Ири...	г.Москва, ул.А...	+79776543217	romanooova@...
14	14	Зайцев Андрей...	г.Москва, ул.Т...	+79776543218	zaycev@gmail...
15	15	Михайлова На...	г.Москва, ул.Л...	+79776543219	mikhailova@m...
16	16	Федорова Анн...	г.Москва, ул.Н...	+79776543220	fedoroova@y...
17	17	Григорьев Мак...	г.Москва, ул.Б...	+79776543221	grigoriiev@gm...
18	18	Козлова Мари...	г.Москва, ул.С...	+79776543222	kozlova@ya...
19	19	Синицын Паве...	г.Москва, ул.П...	+79776543223	sinicyyn@mail...
20	20	Александрова ...	г.Москва, ул.Л...	+79776543224	aleksandroova...
21	21	Ершов Дмитри...	г.Москва, ул.Б...	+79776543225	ershov@gmail...
22	22	Попова Елена ...	г.Москва, ул.Т...	+79776543226	popova@yande...
23	23	Кузнецов Андр...	г.Москва, ул.Л...	+79776543227	kuznetsoov@m...
24	24	Соловьева Тат...	г.Москва, ул.Г...	+79776543228	soolovyyova@ya...
25	25	Николаев Але...	г.Москва, ул.М...	+79776543229	nikolaeev@gm...
26	26	Смирнова Ири...	г.Москва, ул.М...	+79776543230	smirnoova@m...
27	27	Васильева Анн...	г.Москва, ул.А...	+79776543231	vasiljeva@mail.ru
28	28	Иванов Сергей...	г.Москва, ул.Т...	+79176543232	iivanov@gmai...
29	29	Петрова Татя...	г.Москва, ул.Л...	+79776543233	petrova@yande...
30	30	Соколов Дмйт...	г.Москва, ул.Г...	+79776543234	sokolov@amail...

Рис. А.1. Таблица «Клиент» с заполненными данными

	Идентификац...	Id_клиента	Имя	Вид	Порода	Дата_рождения	Пол
1	1	1	Борман	Кот	Шотландский ...	2009-03-01	м
2	2	2	Белка	Собака	Йоркширский ...	2021-05-20	ж
3	3	3	Рыжик	Кот	Сиамская	2022-08-10	м
4	4	4	Жучка	Собака	Немецкая ов...	2021-10-15	ж
5	5	5	Муся	Кошка	Домус	2019-01-04	ж
6	6	6	Барсик	Кот	Сибирский	2018-07-15	м
7	7	7	Лайма	Собака	Мопс	2020-11-02	ж
8	8	8	Пушок	Кот	Персидский	2021-03-28	м
9	9	9	Тоша	Собака	Джек-рассел-т...	2019-09-10	м
10	10	10	Зося	Кошка	Шотландская в...	2022-01-17	ж
11	11	11	Рекс	Собака	Лабрадор-рет...	2020-05-05	м
12	12	12	Мася	Кошка	Британская	2019-12-24	ж
13	13	13	Шарик	Собака	Сибирская хас...	2021-02-14	м
14	14	14	Келла	Кошка	Персидская	2022-04-08	ж
15	15	15	Джек	Собака	Бигль	2019-08-22	м
16	16	16	Рыжик	Кот	Шотландская в...	2020-10-11	м
17	17	17	Белка	Собака	Пудель	2021-06-01	ж
18	18	18	Кузя	Кот	Сиамская	2022-02-20	м
19	19	19	Лёва	Собака	Ротвейлер	2019-12-15	м
20	20	20	Киса	Кошка	Британская	2020-09-07	ж
21	21	21	Бим	Собака	Золотистый ре...	2021-05-28	м
22	22	22	Лелли	Кошка	Шотландская в...	2022-03-14	ж
23	23	23	Чарли	Собака	Боксер	2020-01-18	м
24	24	24	Пушистик	Кот	Персидский	2021-07-02	м
25	25	25	Филя	Кошка	Британская	2022-04-25	ж
26	26	26	Барон	Собака	Немецкая ов...	2019-11-10	м
27	27	27	Васька	Кот	Шотландская п...	2020-08-01	м
28	28	28	Снежка	Кошка	Сиамская	2021-03-15	ж
29	29	29	Боня	Собака	Йоркширский ...	2022-01-28	ж
30	30	30	Баос	Кот	Британская	2020-06-12	м

Рис. А.2. Таблица «Животное» с заполненными данными

	Номер_врача	ФИО	Дата_рождения	Телефон	Должность_С...	Зарплата
1	1	Козлов Алекса...	1985-01-01	+79123457865	Хирург	80000
2	2	Смирнова Юл...	1990-07-07	+79261348969	Терапевт	65000
3	3	Смирнов Дени...	1989-10-29	+79758463793	Ортопед	75000
4	4	Андреева Елиз...	1980-11-06	+79269264829	Дерматолог	70000
5	5	Егоров Андрей...	1975-12-11	+79128482943	Хирург	80000
6	6	Иванов Иван ...	1980-05-10	+79261234567	Терапевт	70000
7	7	Петрова Мари...	1985-08-25	+79775556677	Хирург	85000
8	8	Сидоров Серге...	1992-02-18	+79127890123	Ортопед	75000
9	9	Козлова Анна ...	1988-11-04	+79264567890	Дерматолог	65000
10	10	Петров Дмитр...	1978-03-12	+79121234567	Хирург	60000
11	11	Кузнецова Оль...	1990-06-01	+79771112233	Терапевт	60000
12	12	Соловьёв Ива...	1985-01-01	+79123457000	Терапевт	60000

Рис. А.3. Таблица «Ветеринарный врач» с заполненными данными

	Id_посещения	Идентификац...	Номер_врача	Дата	Диагноз	Итоговая_сто...
1	1	1	1	2023-03-12	Аллергия	5200
2	2	2	2	2023-03-13	Растяжение	1450
3	3	3	3	2023-03-14	Остеохондроз	1400
4	4	4	4	2023-03-17	Аллергия	3091
5	5	5	5	2023-03-25	Гематома	4746
6	6	6	6	2023-03-26	Простуда	700
7	7	7	7	2023-03-27	Перелом лапы	1500
8	8	8	8	2023-03-28	Конъюнктивит	1500
9	9	9	9	2023-03-29	Дерматит	1500
10	10	10	11	2023-03-30	Глисты	6100
11	11	11	11	2023-03-31	Простуда	500
12	12	12	1	2023-04-01	Отит	1000
13	13	13	2	2023-04-02	Растяжение	1000
14	14	14	3	2023-04-03	Остеохондроз	1000
15	15	15	4	2023-04-04	Аллергия	1500
16	16	16	5	2023-04-05	Гематома	2000
17	17	17	6	2023-04-06	Простуда	0
18	18	18	7	2023-04-07	Перелом лапы	1500
19	19	19	8	2023-04-08	Конъюнктивит	1500
20	20	20	9	2023-04-09	Дерматит	1000
21	21	21	6	2023-04-10	Глисты	700
22	22	22	11	2023-04-11	Простуда	0
23	23	23	1	2023-04-12	Отит	2000
24	24	24	2	2023-04-13	Растяжение	700
25	25	25	3	2023-04-14	Остеохондроз	1500
26	26	26	4	2023-04-15	Аллергия	1500
27	27	27	5	2023-04-16	Гематома	700
28	28	28	6	2023-04-17	Простуда	500
29	29	29	7	2023-04-18	Перелом лапы	2000
30	30	30	8	2023-04-19	Конъюнктивит	2000

Рис. А.4. Таблица «Посещение» с заполненными данными

	Номер_услуги	Название_усл...	Стоимость
1	1	Хирургия	1510
	2	Вакцинация	1500
	3	Терапия	700
	4	Анализы	500
	5	Бесплатный ос...	0
	6	Стерилизация	2500
	7	Кастрация	2000
	8	УЗИ	1000
	9	Рентген	1000
	10	Чистка зубов	1200
	11	Стрижка	2500

Рис. А.5. Таблица «Услуга» с заполненными данными

Номер_препа...	Название	Производитель	Количество	Единица_изме...	Цена
1	Антибиотик	Фарма	100	шт	30000
2	Противопараз...	Ветерина	50	шт	15000
3	Витамины	Зоомед	200	шт	25000
4	Анальгетик	Зоомед	75	шт	15000
5	Противовоспа...	Фарма	100	шт	15000
6	Противогрибк...	Ветерина	100	шт	18000
7	Антигельминт...	Фарма	50	шт	15000
8	Пребиотик	Зоомед	200	шт	20000
9	Сердечный пр...	Фарма	50	шт	20000
10	Иммуностиму...	Ветерина	100	шт	24000
11	TestDrug	Test	14	шт	42000

Рис. А.6. Таблица «Медицинский препарат» с заполненными данными

Номер_корма	Название	Производитель	Тип_корма	Количество	Единица_изме...	Цена
1	Royal Canin	Royal Canin	Сухой	277	кг	692500
2	Purina Pro Plan	Purina	Влажный	20	кг	10000
3	Hills Science Diet	Hills	Сухой	20	кг	15000
4	Eukanuba	Eukanuba	Сухой	30	кг	20000
5	Acana	Acana	Сухой	25	кг	23000
6	Brit Care	Brit Care	Сухой	40	кг	30000
7	Orijen	Orijen	Сухой	20	кг	10000
8	Go! Natural	Go!	Сухой	30	кг	20000
9	Nutra Nuggets	Nutra Nuggets	Влажный	15	кг	12000
10	Whiskas	Whiskas	Влажный	25	шт	3000
11	Test	Test	Сухой	14	кг	49000

Рис. А.7. Таблица «Корм» с заполненными данными

Id_расхода	Id_посещения
1	1
2	2
3	3
4	4
5	5
6	10
13	13
11	16
10	27
7	37
8	50
9	52
12	78
14	97
15	103
16	105

Рис. А.8. Таблица «Расход» с заполненными данными

Номер_закупки	Дата_закупки	Поставщик	Дата_поставки	Итого
1	2023-03-01	Фарма	2023-03-05	26000
2	2023-03-10	Ветерина	2023-03-15	9666
3	2023-03-20	Зоомед	2023-03-25	10000
4	2023-03-25	Зоомед	2023-03-30	18000
5	2023-03-30	Фарма	2023-04-05	22000
6	2023-04-05	Фарма	2023-04-10	7000
7	2023-04-10	Ветерина	2023-04-15	7000
8	2023-04-15	Зоомед	2023-04-20	12000
9	2023-04-20	Фарма	2023-04-25	18000
10	2023-04-25	Ветерина	2023-04-30	10000
11	2023-04-30	Зоомед	2023-05-05	10000
12	2023-05-05	Фарма	2023-05-10	5000
13	2023-05-10	Ветерина	2023-05-15	19000
14	2023-05-15	Зоомед	2023-05-20	13000
15	2023-05-20	Фарма	2023-05-25	8000
16	2023-05-25	Ветерина	2023-05-30	12000
17	2023-05-30	Зоомед	2023-06-05	6000
18	2023-06-05	Фарма	2023-06-10	16000
19	2023-06-10	Ветерина	2023-06-15	10000
20	2023-06-15	Зоомед	2023-06-20	8000
21	2023-06-20	Фарма	2023-06-25	10000
22	2023-06-25	Ветерина	2023-06-30	18000
23	2023-06-30	Зоомед	2023-07-05	8000
24	2023-07-05	Фарма	2023-07-10	14000
25	2023-07-10	Ветерина	2023-07-15	12000
26	2023-07-15	Зоомед	2023-07-20	6000
27	2023-07-20	Фарма	2023-07-25	10000
28	2023-07-25	Ветерина	2023-07-30	4000
29	2023-07-30	Зоомед	2023-08-05	13000
30	2023-08-05	Фарма	2023-08-10	12000

Рис. А.9. Таблица «Закупка» с заполненными данными

	Id_посещения	Номер_услуги
▶	1	1
	1	3
	1	4
	1	5
	2	3
	2	5
	3	3
	4	2
	4	4
	5	1
	5	2
	6	3
	7	1
	8	2
	9	2
	10	4
	11	4
	12	9
	13	9
	14	9
	15	2
	16	1
	17	5
	18	1
	19	2
	20	3
	21	3
	22	5
	23	1
	24	3

1 для 110

Рис. А.10. Таблица «ПосещениеУслуга» с заполненными данными

	Id_расхода	Номер_препа...	Количество
▶	1	1	1
	3	4	1
	4	2	1
	4	3	1
	5	5	1
	6	2	2
	7	3	1
	8	5	3
	9	5	1
	10	4	2
	11	5	2
	12	1	3
	13	4	1
	14	7	1
	16	11	1

Рис. А.11. Таблица «РасходПрепарат» с заполненными данными

	Id_расхода	Номер_корма	Количество
1	1	1	1
2	3	1	1
3	2	1	1
4	4	1	1
5	4	1	1
5	5	1	1
6	1	2	2
7	3	2	2
8	4	1	1
9	5	2	2
10	6	2	2
11	2	3	3
12	2	3	3
13	7	2	2
14	8	2	2
16	11	1	1

Рис. А.12. Таблица «РасходКорм» с заполненными данными

	Номер_закупки	Номер_препа...	Количество
1	1	10	10
1	2	10	10
2	4	15	15
3	3	15	15
4	4	20	20
5	5	10	10
6	5	15	15
7	6	18	18
8	3	10	10
9	9	12	12
10	2	16	16
11	8	10	10
12	7	15	15
13	10	12	12
14	4	18	18
15	1	14	14
16	6	12	12
17	3	16	16
18	1	10	10
19	6	14	14
20	8	18	18
21	7	12	12
22	10	10	10
23	8	16	16
24	9	12	12
25	10	18	18
26	4	14	14
27	1	10	10
28	6	18	18
29	4	16	16

Рис. А.13. Таблица «ЗакупкаПепарат» с заполненными данными

Номер_закупки	Номер_корма	Количество
1	1	5
1	2	15
2	4	10
3	3	10
4	4	20
5	5	10
6	1	15
6	3	10
7	2	12
7	4	18
8	5	16
8	6	14
9	7	10
9	8	12
10	9	18
10	10	16
11	1	14
11	2	10
12	3	18
12	4	12
13	5	16
13	6	10
14	7	14
14	8	18
15	9	12
15	10	16
16	1	10
16	2	14
17	3	18
17	4	12

Рис. А.14. Таблица «ЗакупкаКорм» с заполненными данными

SQL-КОМАНДЫ СОЗДАНИЯ ПРОГРАММНЫХ ОБЪЕКТОВ БД

1. Хранимая процедура AddClient

```

ALTER PROCEDURE [dbo].[AddClient] (
    @ФИО VARCHAR(50),
    @Адрес VARCHAR(80),
    @Телефонный_номер VARCHAR(20),
    @Электронная_почта VARCHAR(50))
AS
BEGIN
    DECLARE @NewId INT;
    -- Получаем максимальный Id_клиента и увеличиваем на 1
    SELECT @NewId = ISNULL(MAX(Id_клиента), 0) + 1 FROM Клиент;
    -- Проверка на существование клиента с таким телефонным номером
или email
    IF EXISTS (SELECT * FROM Клиент WHERE Телефонный_номер =
@Телефонный_номер)
    BEGIN
        RAISERROR('Клиент с таким телефонным номером уже
существует.', 16, 1);
        RETURN;
    END
    IF EXISTS (SELECT * FROM Клиент WHERE Электронная_почта =
@Электронная_почта)
    BEGIN
        RAISERROR('Клиент с таким email уже существует.', 16, 1);
        RETURN;
    END
    -- Вставка нового клиента в таблицу

```

```
INSERT INTO Клиент (Id_клиента, ФИО, Адрес, Телефонный_номер,  
Электронная_почта)
```

```
VALUES (@NewId, @ФИО, @Адрес, @Телефонный_номер,  
@Электронная_почта);
```

```
END;
```

2. Хранимая процедура AddAnimal

```
ALTER PROCEDURE [dbo].[AddAnimal] (
```

```
@Id_клиента INT,
```

```
@Имя VARCHAR(10),
```

```
@Вид VARCHAR(10),
```

```
@Порода VARCHAR(25),
```

```
@Дата_рождения DATE,
```

```
@Пол VARCHAR(1),
```

```
@ErrorMessage VARCHAR(255) OUTPUT -- Добавлена выходная  
переменная
```

```
)
```

```
AS
```

```
BEGIN
```

```
SET @ErrorMessage = NULL; -- Инициализация переменной
```

```
-- Проверка на корректность данных клиента
```

```
IF (SELECT COUNT(*) FROM Клиент WHERE Id_клиента =  
@Id_клиента) = 0
```

```
BEGIN
```

```
SET @ErrorMessage = 'Такого клиента не существует.';
```

```
RETURN;
```

```
END
```

```
-- Проверка даты рождения: дата не может быть в будущем
```

```
IF @Дата_рождения > GETDATE()
```

```
BEGIN
```

```
SET @ErrorMessage = 'Дата рождения не может быть в будущем.';
```

```

        RETURN;
    END
    -- Генерация нового идентификационного номера (автоматически)
    DECLARE @NextId INT;
    SELECT @NextId = ISNULL(MAX(Идентификационный_номер), 0) +
1 FROM Животное;
    -- Вставка нового животного
    INSERT INTO Животное (Идентификационный_номер, Id_клиента,
Имя, Вид, Порода, Дата_рождения, Пол)
        VALUES (@NextId, @Id_клиента, @Имя, @Вид, @Порода,
@Дата_рождения, @Пол);
    END;
3. Хранимая процедура AddVisit
ALTER PROCEDURE [dbo].[AddVisit] (
    @Идентификационный_номер INT,
    @Номер_врача INT,
    @Дата DATE,
    @Диагноз VARCHAR(25),
    @Итоговая_стоимость INT,
    @ErrorMessage VARCHAR(255) OUTPUT -- Добавляем выходную
переменную для сообщений об ошибке
)
AS
BEGIN
    SET @ErrorMessage = NULL; -- Инициализация переменной
    -- Проверка на существование животного
    IF (SELECT COUNT(*) FROM Животное WHERE
Идентификационный_номер = @Идентификационный_номер) = 0
        BEGIN

```

```

        SET @ErrorMessage = 'Животное с таким идентификационным
номером не существует.';

        RETURN;

    END

    -- Проверка на существование врача
    IF (SELECT COUNT(*) FROM Ветеринарный_врач WHERE
Номер_врача = @Номер_врача) = 0

    BEGIN

        SET @ErrorMessage = 'Врач с таким номером не существует.';

        RETURN;

    END

    -- Проверка на корректность даты
    IF @Дата > GETDATE()

    BEGIN

        SET @ErrorMessage = 'Дата посещения не может быть в будущем.';

        RETURN;

    END

    -- Генерация нового идентификатора посещения (автоматически)
    DECLARE @NextId INT;

    SELECT @NextId = ISNULL(MAX(Id_посещения), 0) + 1 FROM
Посещение;

    -- Вставка нового посещения

    INSERT INTO Посещение (Id_посещения,
Идентификационный_номер, Номер_врача, Дата, Диагноз,
Итоговая_стоимость)

        VALUES (@NextId, @Идентификационный_номер, @Номер_врача,
@Дата, @Диагноз, @Итоговая_стоимость);

    -- Возвращаем сгенерированный Id_посещения
    SELECT @NextId AS Id_посещения;

    END

```

4. Хранимая процедура AddVetDoctor

```
ALTER PROCEDURE [dbo].[AddVetDoctor] (  
    @ФИО VARCHAR(50),  
    @Дата_рождения DATE,  
    @Телефон VARCHAR(20),  
    @Должность_Специальность VARCHAR(25),  
    @Зарплата INT,  
    @ErrorMessage VARCHAR(255) OUTPUT -- Добавляем выходную  
переменную для сообщений об ошибке  
)  
AS  
BEGIN  
    SET @ErrorMessage = NULL; -- Инициализация переменной  
    -- Проверка на возраст врача  
    IF DATEDIFF(YEAR, @Дата_рождения, GETDATE()) < 18  
    BEGIN  
        SET @ErrorMessage = 'Врач должен быть старше 18 лет.';  
        RETURN;  
    END  
    -- Проверка на минимальную зарплату  
    IF @Зарплата <= 30000  
    BEGIN  
        SET @ErrorMessage = 'Зарплата должна быть больше 30000.';  
        RETURN;  
    END  
    -- Генерация нового номера врача (автоматически)  
    DECLARE @NextId INT;  
    SELECT @NextId = ISNULL(MAX(Номер_врача), 0) + 1 FROM  
Ветеринарный_врач;  
    -- Вставка нового врача
```



```

        INSERT INTO Ветеринарный_врач (Номер_врача, ФИО,
Дата_рождения, Телефон, Должность_Специальность, Зарплата)
        VALUES (@NextId, @ФИО, @Дата_рождения, @Телефон,
@Должность_Специальность, @Зарплата);
        -- Возвращаем сгенерированный Номер_врача
        SELECT @NextId AS Номер_врача;
END

5. Триггер trgCheckServiceCost
ALTER TRIGGER [dbo].[trgCheckServiceCost]
ON [dbo].[Услуга]
INSTEAD OF INSERT
AS
BEGIN
    DECLARE @Стоимость INT, @Название_услуги VARCHAR(25),
@Номер_услуги INT;
    -- Извлечение данных из вставляемых строк
    SELECT @Стоимость = Стоимость, @Название_услуги =
Название_услуги FROM inserted;
    -- Проверка на стоимость
    IF @Стоимость < 0 AND @Название_услуги != 'Осмотр'
    BEGIN
        RAISERROR('Стоимость услуги должна быть больше 0, за
исключением бесплатных осмотров.', 16, 1);
        RETURN;
    END
    -- Получаем максимальный номер услуги и увеличиваем на 1
    SELECT @Номер_услуги = ISNULL(MAX(Номер_услуги), 0) + 1
FROM dbo.Услуга;

```

```

-- Вставка данных в таблицу Услуга с автоматически генерируемым
Номер_услуги
INSERT INTO Услуга (Номер_услуги, Название_услуги, Стоимость)
SELECT  @Номер_услуги,  Название_услуги,  Стоимость  FROM
inserted;
END;

```

6. Хранимая процедура AddFood

```

ALTER PROCEDURE [dbo].[AddFood] (
    @Название VARCHAR(25),
    @Производитель VARCHAR(25),
    @Тип_корма VARCHAR(25),
    @Количество INT,
    @Единица_измерения VARCHAR(5),
    @Цена INT,
    @ErrorMessage VARCHAR(255) OUTPUT, -- Параметр для ошибок
    @SuccessMessage VARCHAR(255) OUTPUT -- Параметр для
успешного сообщения
)
AS
BEGIN
    SET @ErrorMessage = NULL; -- Инициализация переменной для
ошибок
    SET @SuccessMessage = NULL; -- Инициализация переменной для
успешного сообщения
    -- Проверка на количество > 0
    IF @Количество <= 0
    BEGIN
        SET @ErrorMessage = 'Ошибка: Количество должно быть больше
нуля.';
    END
    RETURN;

```

```

END
-- Проверка на цену > 0
IF @Цена <= 0
BEGIN
    SET @ErrorMessage = 'Ошибка: Цена должна быть больше нуля.';
    RETURN;
END
DECLARE @NewНомер_корма INT;
-- Получаем максимальный Номер_корма и увеличиваем на 1
SELECT @NewНомер_корма = ISNULL(MAX(Номер_корма), 0) + 1
FROM Корм;
-- Вставка нового корма с автоматически сгенерированным
Номером_корма
INSERT INTO Корм (Номер_корма, Название, Производитель,
Тип_корма, Количество, Единица_измерения, Цена)
VALUES (@NewНомер_корма, @Название, @Производитель,
@Тип_корма, @Количество, @Единица_измерения, @Цена);
-- Если все прошло успешно, устанавливаем сообщение об успехе
SET @SuccessMessage = 'Данные успешно добавлены в таблицу
Корм.';
END;

```

7. Хранимая процедура AddMedicalDrug

```

ALTER PROCEDURE [dbo].[AddMedicalDrug] (
    @Название VARCHAR(25),
    @Производитель VARCHAR(25),
    @Количество INT,
    @Единица_измерения VARCHAR(5),
    @Цена INT,
    @ErrorMessage VARCHAR(255) OUTPUT, -- Параметр для ошибок

```

```

        @SuccessMessage VARCHAR(255) OUTPUT    -- Параметр для
успешного сообщения
    )
AS
BEGIN
    SET @ErrorMessage = NULL; -- Инициализация переменной для
ошибок
    SET @SuccessMessage = NULL; -- Инициализация переменной для
успешного сообщения

    -- Проверка на количество > 0
    IF @Количество <= 0
    BEGIN
        SET @ErrorMessage = 'Ошибка: Количество должно быть больше
нуля.';

        RETURN;
    END

    -- Проверка на цену > 0
    IF @Цена <= 0
    BEGIN
        SET @ErrorMessage = 'Ошибка: Цена должна быть больше нуля.';
        RETURN;
    END

    DECLARE @NewНомер_препарата INT;

    -- Получаем максимальный Номер_препарата и увеличиваем на 1
    SELECT @NewНомер_препарата = ISNULL(MAX(Номер_препарата),
0) + 1 FROM Медицинский_препарат;

    -- Вставка нового медицинского препарата с автоматически
сгенерированным Номером_препарата
    INSERT INTO Медицинский_препарат (Номер_препарата, Название,
Производитель, Количество, Единица_измерения, Цена)

```

```
VALUES (@NewНомер_препарата, @Название, @Производитель,
@Количество, @Единица_измерения, @Цена);
-- Если все прошло успешно, устанавливаем сообщение об успехе
SET @SuccessMessage = 'Данные успешно добавлены в таблицу
Медицинский препарат.';
END;
```

8. Хранимая процедура AddZakupka

```
ALTER PROCEDURE [dbo].[AddZakupka]
    @Дата_закупки DATE,
    @Поставщик VARCHAR(25),
    @Дата_поставки DATE,
    @Итого INT,
    @ErrorMessage VARCHAR(255) OUTPUT,
    @SuccessMessage VARCHAR(255) OUTPUT
AS
BEGIN
    SET @ErrorMessage = NULL;
    SET @SuccessMessage = NULL;
    -- Проверка на корректность даты поставки
    IF @Дата_поставки <= @Дата_закупки
    BEGIN
        SET @ErrorMessage = 'Ошибка: Дата поставки должна быть позже
даты закупки.';
        RETURN;
    END
    DECLARE @NewНомер_закупки INT;
    -- Получаем максимальный Номер_закупки и увеличиваем на 1
    SELECT @NewНомер_закупки = ISNULL(MAX(Номер_закупки), 0)
+ 1 FROM Закупка;
```

```

-- Вставка новой закупки с автоматически сгенерированным
Номером_закупки
BEGIN TRY
    INSERT INTO Закупка (Номер_закупки, Дата_закупки, Поставщик,
Дата_поставки, Итого)
        VALUES (@NewНомер_закупки, @Дата_закупки, @Поставщик,
@Дата_поставки, @Итого);
    SET @SuccessMessage = 'Закупка успешно добавлена.';
END TRY
BEGIN CATCH
    SET @ErrorMessage = 'Ошибка при добавлении закупки: ' +
ERROR_MESSAGE();
END CATCH
END;

```

9. Хранимая процедура AddZakupkaFood

```

ALTER PROCEDURE [dbo].[AddZakupkaFood]
    @Номер_закупки INT,
    @Номер_корма INT,
    @Количество INT,
    @ErrorMessage VARCHAR(255) OUTPUT,
    @SuccessMessage VARCHAR(255) OUTPUT
AS
BEGIN
    SET @ErrorMessage = NULL;
    SET @SuccessMessage = NULL;
    -- Проверка на существование закупки
    IF NOT EXISTS (SELECT * FROM Закупка WHERE Номер_закупки
= @Номер_закупки)
        BEGIN

```

```

        SET @ErrorMessage = 'Ошибка: Указанный номер закупки не
существует.';

        RETURN;

    END

    -- Проверка на существование корма
    IF NOT EXISTS (SELECT* FROM Корм WHERE Номер_корма =
@Номер_корма)
    BEGIN

        SET @ErrorMessage = 'Ошибка: Указанный номер корма не
существует.';

        RETURN;

    END

    -- Проверка на количество
    IF @Количество <= 0
    BEGIN

        SET @ErrorMessage = 'Ошибка: Количество должно быть больше
нуля.';

        RETURN;

    END

    -- Вставка записи о закупке корма
    BEGIN TRY

        INSERT INTO ЗакупкаКорм (Номер_закупки, Номер_корма,
Количество)

        VALUES (@Номер_закупки, @Номер_корма, @Количество);

        SET @SuccessMessage = 'Закупка корма успешно добавлена.';

    END TRY

    BEGIN CATCH

        SET @ErrorMessage = 'Ошибка при добавлении закупки корма: ' +
ERROR_MESSAGE();

    END CATCH

```

END;

10. Хранимая процедура AddZakupkaDrug

ALTER PROCEDURE [dbo].[AddZakupkaDrug]

 @Номер_закупки INT,

 @Номер_препарата INT,

 @Количество INT,

 @ErrorMessage VARCHAR(255) OUTPUT,

 @SuccessMessage VARCHAR(255) OUTPUT

AS

BEGIN

 SET @ErrorMessage = NULL;

 SET @SuccessMessage = NULL;

 -- Проверка на существование закупки

 IF NOT EXISTS (SELECT * FROM Закупка WHERE Номер_закупки
= @Номер_закупки)

 BEGIN

 SET @ErrorMessage = 'Ошибка: Указанный номер закупки не
существует.';

 RETURN;

 END

 -- Проверка на существование препарата

 IF NOT EXISTS (SELECT * FROM Медицинский_препарат WHERE
Номер_препарата = @Номер_препарата)

 BEGIN

 SET @ErrorMessage = 'Ошибка: Указанный номер препарата не
существует.';

 RETURN;

 END

 -- Проверка на количество

 IF @Количество <= 0


```

BEGIN
    SET @ErrorMessage = 'Ошибка: Количество должно быть больше
нуля.';
    RETURN;
END
-- Вставка записи о покупке препарата
BEGIN TRY
    INSERT INTO ЗакупкаПрепарат (Номер_закупки,
Номер_препарата, Количество)
    VALUES (@Номер_закупки, @Номер_препарата, @Количество);
    SET @SuccessMessage = 'Закупка препарата успешно добавлена.';
END TRY
BEGIN CATCH
    SET @ErrorMessage = 'Ошибка при добавлении закупки препарата:
' + ERROR_MESSAGE();
END CATCH
END;

```

11. Хранимая процедура AddRashod

```

ALTER PROCEDURE [dbo].[AddRashod]
    @Id_посещения INT, -- Мы передаем только Id_посещения
    @ErrorMessage VARCHAR(255) OUTPUT,
    @SuccessMessage VARCHAR(255) OUTPUT
AS
BEGIN
    SET @ErrorMessage = NULL;
    SET @SuccessMessage = NULL;
    -- Проверка на существование записи с таким Id_посещения
    IF NOT EXISTS (SELECT * FROM Посещение WHERE
Id_посещения = @Id_посещения)
    BEGIN

```

```

        SET @ErrorMessage = 'Ошибка: Указанное посещение не
существует.';

        RETURN;

    END

    DECLARE @NewId_расхода INT;

    -- Получаем максимальный Id_расхода и увеличиваем на 1
    SELECT @NewId_расхода = ISNULL(MAX(Id_расхода), 0) + 1
    FROM Расход;

    -- Вставка нового расхода с автоматически сгенерированным
    Id_расхода

    BEGIN TRY

        INSERT INTO Расход (Id_расхода, Id_посещения)
        VALUES (@NewId_расхода, @Id_посещения);

        SET @SuccessMessage = 'Расход успешно добавлен.';

    END TRY

    BEGIN CATCH

        SET @ErrorMessage = 'Ошибка при добавлении расхода: ' +
        ERROR_MESSAGE();

    END CATCH

END;

```

12. Хранимая процедура AddRashodKorm

```
ALTER PROCEDURE [dbo].[AddRashodKorm]
```

```

    @Id_расхода INT,
    @Номер_корма INT,
    @Количество INT,
    @ErrorMessage VARCHAR(255) OUTPUT,
    @SuccessMessage VARCHAR(255) OUTPUT

```

```
AS
```

```
BEGIN
```

```
    SET @ErrorMessage = NULL;
```

```

SET @SuccessMessage = NULL;

-- Проверка на существование расхода
IF NOT EXISTS (SELECT * FROM Расход WHERE Id_расхода =
@Id_расхода)
BEGIN
    SET @ErrorMessage = 'Ошибка: Указанный Id расхода не
существует.';
    RETURN;
END

-- Проверка на существование корма
IF NOT EXISTS (SELECT * FROM Корм WHERE Номер_корма =
@Номер_корма)
BEGIN
    SET @ErrorMessage = 'Ошибка: Указанный номер корма не
существует.';
    RETURN;
END

-- Проверка на количество
IF @Количество <= 0
BEGIN
    SET @ErrorMessage = 'Ошибка: Количество должно быть больше
нуля.';
    RETURN;
END

-- Вставка записи о расходе корма
BEGIN TRY
    INSERT INTO РасходКорм (Id_расхода, Номер_корма, Количество)
    VALUES (@Id_расхода, @Номер_корма, @Количество);
    SET @SuccessMessage = 'Запись о расходе корма успешно
добавлена.';

```

```

END TRY
BEGIN CATCH
    SET @ErrorMessage = 'Ошибка при добавлении расхода корма: ' +
ERROR_MESSAGE();
END CATCH
END;

```

13. Хранимая процедура AddRashodPreparat

```

ALTER PROCEDURE [dbo].[AddRashodPreparat]
    @Id_расхода INT,
    @Номер_препарата INT,
    @Количество INT,
    @ErrorMessage VARCHAR(255) OUTPUT,
    @SuccessMessage VARCHAR(255) OUTPUT
AS
BEGIN
    SET @ErrorMessage = NULL;
    SET @SuccessMessage = NULL;
    -- Проверка на существование расхода
    IF NOT EXISTS (SELECT * FROM Расход WHERE Id_расхода =
@Id_расхода)
    BEGIN
        SET @ErrorMessage = 'Ошибка: Указанный Id расхода не
существует.';
        RETURN;
    END
    -- Проверка на существование медицинского препарата
    IF NOT EXISTS (SELECT * FROM Медицинский_препарат WHERE
Номер_препарата = @Номер_препарата)
    BEGIN

```

```

        SET @ErrorMessage = 'Ошибка: Указанный номер препарата не
существует.';

        RETURN;

    END

    -- Проверка на количество
    IF @Количество <= 0
    BEGIN

        SET @ErrorMessage = 'Ошибка: Количество должно быть больше
нуля.';

        RETURN;

    END

    -- Вставка записи о расходе препарата
    BEGIN TRY

        INSERT INTO РасходПрепарат (Id_расхода, Номер_препарата,
Количество)

        VALUES (@Id_расхода, @Номер_препарата, @Количество);

        SET @SuccessMessage = 'Запись о расходе препарата успешно
добавлена.';

    END TRY

    BEGIN CATCH

        SET @ErrorMessage = 'Ошибка при добавлении расхода препарата:
' + ERROR_MESSAGE();

    END CATCH

END;

14. Хранимая процедура Рассчитать_Итоговую_Сумму_Закупки
ALTER PROCEDURE [dbo].[Рассчитать_Итоговую_Сумму_Закупки]
    @Номер_закупки INT
AS
BEGIN

    DECLARE @Итоговая_Сумма INT = 0;

```

```

-- Рассчитываем сумму для закупки корма
DECLARE @КормСумма INT;
SELECT @КормСумма = SUM(zk.Количество * k.Цена /
k.Количество) -- Цена делится на Количество в таблице Корм, чтобы учесть
цену за единицу
FROM ЗакупкаКорм zk
JOIN Корм k ON zk.Номер_корма = k.Номер_корма
WHERE zk.Номер_закупки = @Номер_закупки;
-- Если сумма для корма найдена, добавляем ее к итоговой сумме
SET @Итоговая_Сумма = @Итоговая_Сумма +
COALESCE(@КормСумма, 0);
-- Рассчитываем сумму для закупки медицинских препаратов
DECLARE @ПрепаратСумма INT;
SELECT @ПрепаратСумма = SUM(zp.Количество * mp.Цена /
mp.Количество) -- Цена делится на Количество в таблице Медицинский
препарат, чтобы учесть цену за единицу
FROM ЗакупкаПрепарат zp
JOIN Медицинский_препарат mp ON zp.Номер_препарата =
mp.Номер_препарата
WHERE zp.Номер_закупки = @Номер_закупки;
-- Если сумма для препарата найдена, добавляем ее к итоговой сумме
SET @Итоговая_Сумма = @Итоговая_Сумма +
COALESCE(@ПрепаратСумма, 0);
-- Обновляем поле "Итого" в таблице Закупка
UPDATE Закупка
SET Итого = @Итоговая_Сумма
WHERE Номер_закупки = @Номер_закупки;
-- Возвращаем итоговую сумму
SELECT @Итоговая_Сумма AS Итоговая_Сумма_Закупки;
END;

```

15. Хранимая процедура Рассчитать_Общее_Количество_Корма

ALTER PROCEDURE [dbo].[Рассчитать_Общее_Количество_Корма]

 @Номер_корма INT

AS

BEGIN

 DECLARE @Общее_Количество INT;

 -- Изначальное количество (по умолчанию - текущее в таблице Корм)

 SELECT @Общее_Количество = Количество

 FROM Корм

 WHERE Номер_корма = @Номер_корма;

 -- Прибавляем количество из ЗакупкаКорм

 SELECT @Общее_Количество = @Общее_Количество +

COALESCE(SUM(Количество), 0)

 FROM ЗакупкаКорм

 WHERE Номер_корма = @Номер_корма;

 -- Уменьшаем количество из РасходКорм

 SELECT @Общее_Количество = @Общее_Количество -

COALESCE(SUM(Количество), 0)

 FROM РасходКорм

 WHERE Номер_корма = @Номер_корма;

 -- Возвращаем результат

 SELECT @Общее_Количество AS Общее_Количество_Корма;

END;

16. Хранимая процедура Рассчитать_Сумму_Корма

ALTER PROCEDURE [dbo].[Рассчитать_Сумму_Корма]

 @Номер_корма INT -- Номер корма

AS

BEGIN

 DECLARE @Цена INT; -- Цена за единицу корма

 DECLARE @Количество INT; -- Текущее количество корма

```

DECLARE @Цена_за_кг INT; -- Цена за 1 кг корма
DECLARE @Общее_Количество INT; -- Общее количество корма с
учетом закупки и расхода

DECLARE @Расход INT;    -- Общий расход корма (если есть)
DECLARE @Сумма INT;     -- Итоговая сумма

-- Получаем цену и количество для указанного корма
SELECT @Цена = Цена, @Количество = Количество
FROM Корм
WHERE Номер_корма = @Номер_корма;

-- Проверка на наличие данных по корму
IF @Цена IS NULL OR @Количество IS NULL
BEGIN
    SET @Сумма = 0;
    SELECT @Сумма AS Сумма_Корма;
    RETURN;
END

-- Получаем расход (если есть) из таблицы РасходКорм
SELECT @Расход = COALESCE(SUM(Количество), 0)
FROM РасходКорм
WHERE Номер_корма = @Номер_корма;

-- Учитываем закупки из таблицы ЗакупкаКорм
SELECT      @Общее_Количество      =      @Количество      +
COALESCE(SUM(Количество), 0)

FROM ЗакупкаКорм
WHERE Номер_корма = @Номер_корма;

-- Теперь пересчитываем цену за 1 кг
-- Цена за 1 кг рассчитывается как (Цена / Количество)
SET @Цена_за_кг = @Цена / @Количество;

-- Пересчитываем цену с учетом закупки и расхода
SET @Сумма = @Цена_за_кг * (@Общее_Количество - @Расход);

```



```

-- Возвращаем результат
SELECT @Сумма AS Сумма_Корма, @Общее_Количество AS
Общее_Количество;
END;

17. Хранимая процедура Рассчитать_Общее_Количество_Препарата
ALTER PROCEDURE
[dbo].[Рассчитать_Общее_Количество_Препарата]
    @Номер_препарата INT
AS
BEGIN
    DECLARE @Общее_Количество INT;

    -- Изначальное количество (по умолчанию - текущее в таблице
    Медицинский_препарат)
    SELECT @Общее_Количество = Количество
    FROM Медицинский_препарат
    WHERE Номер_препарата = @Номер_препарата;

    -- Прибавляем количество из ЗакупкаПрепарат
    SELECT @Общее_Количество = @Общее_Количество +
COALESCE(SUM(Количество), 0)
    FROM ЗакупкаПрепарат
    WHERE Номер_препарата = @Номер_препарата;

    -- Уменьшаем количество из РасходПрепарат
    SELECT @Общее_Количество = @Общее_Количество -
COALESCE(SUM(Количество), 0)
    FROM РасходПрепарат
    WHERE Номер_препарата = @Номер_препарата;

    -- Возвращаем результат
    SELECT @Общее_Количество AS Общее_Количество_Препарата;
END;

18. Хранимая процедура Рассчитать_Сумму_Препарата

```

```

ALTER PROCEDURE [dbo].[Рассчитать_Сумму_Препарата]
    @Номер_препарата INT -- Номер препарата
AS
BEGIN
    DECLARE @Цена INT;      -- Цена за единицу препарата
    DECLARE @Количество INT; -- Текущее количество препарата
    DECLARE @Общее_Количество INT; -- Общее количество
препарата с учетом закупки и расхода
    DECLARE @Расход INT;      -- Общий расход препарата (если есть)
    DECLARE @Цена_за_штуку INT; -- Цена за одну штуку
    DECLARE @Сумма INT;      -- Итоговая сумма
    -- Получаем цену и количество для указанного препарата
    SELECT @Цена = Цена, @Количество = Количество
    FROM Медицинский_препарат
    WHERE Номер_препарата = @Номер_препарата;
    -- Проверка на наличие данных по препарату
    IF @Цена IS NULL OR @Количество IS NULL
    BEGIN
        SET @Сумма = 0;
        SELECT @Сумма AS Сумма_Препарата;
        RETURN;
    END
    -- Получаем расход (если есть) из таблицы РасходПрепарат
    SELECT @Расход = COALESCE(SUM(Количество), 0)
    FROM РасходПрепарат
    WHERE Номер_препарата = @Номер_препарата;
    -- Учитываем закупки из таблицы ЗакупкаПрепарат
    SELECT      @Общее_Количество      =      @Количество      +
COALESCE(SUM(Количество), 0)
    FROM ЗакупкаПрепарат

```

```

WHERE Номер_препарата = @Номер_препарата;
-- Цена за 1 штуку (Цена / Количество)
SET @Цена_за_штуку = @Цена / @Количество;
-- Пересчитываем цену с учетом закупки и расхода
SET @Сумма = @Цена_за_штуку * (@Общее_Количество -
@Расход);
-- Возвращаем результат
SELECT @Сумма AS Сумма_Препарата, @Общее_Количество AS
Общее_Количество;
END;

```

19. Хранимая процедура РассчитатьИтоговуюСтоимость

```
ALTER PROCEDURE [dbo].[РассчитатьИтоговуюСтоимость]
```

```
    @Id_посещения INT
```

```
AS
```

```
BEGIN
```

```
    DECLARE @ИтоговаяСтоимость INT
```

```
    DECLARE @СтоимостьУслуг INT
```

```
    DECLARE @СтоимостьПрепаратов INT
```

```
    DECLARE @СтоимостьКорма INT
```

```
    -- Инициализация итоговой стоимости
```

```
    SET @ИтоговаяСтоимость = 0
```

```
    -- 1. Рассчитываем стоимость услуг
```

```
    SELECT @СтоимостьУслуг = SUM(Услуга.Стоимость)
```

```
    FROM ПосещениеУслуга
```

```
    JOIN      Услуга      ON      ПосещениеУслуга.Номер_услуги      =
```

```
    Услуга.Номер_услуги
```

```
    WHERE ПосещениеУслуга.Id_посещения = @Id_посещения
```

```
    -- 2. Рассчитываем стоимость израсходованных медицинских
препаратов
```

```

SELECT                                     @СтоимостьПрепаратов
=SUM(Медицинский_препарат.Цена / Медицинский_препарат.Количество *
РасходПрепарат.Количество)
FROM РасходПрепарат
JOIN Медицинский_препарат ON РасходПрепарат.Номер_препарата
= Медицинский_препарат.Номер_препарата
JOIN Расход ON РасходПрепарат.Id_расхода = Расход.Id_расхода
WHERE Расход.Id_посещения = @Id_посещения
-- 3. Рассчитываем стоимость израсходованного корма
SELECT @СтоимостьКорма = SUM(Корм.Цена / Корм.Количество *
РасходКорм.Количество)
FROM РасходКорм
JOIN Корм ON РасходКорм.Номер_корма = Корм.Номер_корма
JOIN Расход ON РасходКорм.Id_расхода = Расход.Id_расхода
WHERE Расход.Id_посещения = @Id_посещения
IF @СтоимостьУслуг IS NULL SET @СтоимостьУслуг = 0
IF @СтоимостьПрепаратов IS NULL SET @СтоимостьПрепаратов = 0
IF @СтоимостьКорма IS NULL SET @СтоимостьКорма = 0
SET      @ИтоговаяСтоимость      =      @СтоимостьУслуг      +
@СтоимостьПрепаратов + @СтоимостьКорма
UPDATE Посещение
SET Итоговая_стоимость = @ИтоговаяСтоимость
WHERE Id_посещения = @Id_посещения
SELECT @ИтоговаяСтоимость AS Итоговая_стоимость
END

```

КОД ПРОГРАММЫ

1. Форма «Начать работу»

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
    }
    private void butvrach_Click(object sender, EventArgs e)
    {
        Form form4 = new Form4();
        form4.Show();
    }
    private void butMedStat_Click(object sender, EventArgs e)
    {
        Form form5= new Form5();
        form5.Show();
    }
    private void butadmin_Click(object sender, EventArgs e)
    {
        Form form3 = new Form3();
        form3.Show();
    }
    private void butMeneg_Click(object sender, EventArgs e)
    {
        Form form6= new Form6();
        form6.Show();
    }
}
```

2. Форма «Администратор»

```
public partial class Form3 : Form
{
    public Form3()
    {
        InitializeComponent();
    }

    private void животноеBindingNavigatorSaveItem_Click(object sender, EventArgs e)
    {
        this.Validate();
        this.животноеBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }

    private void Form3_Load(object sender, EventArgs e)
```

```

        {
            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Ветеринарный_врач". При необходимости она может быть
            перемещена или удалена.

            this.ветеринарный_врачTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Ветеринарный_врач)
            ;

            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Посещение". При необходимости она может быть перемещена
            или удалена.
            this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);
            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Клиент". При необходимости она может быть перемещена или
            удалена.
            this.клиентTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Клиент);
            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Животное". При необходимости она может быть перемещена
            или удалена.
            this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);
        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void клиентDataGridView_CellContentClick(object sender,
        DataGridViewCellEventArgs e)
        {

        }

        private void butsave_Click(object sender, EventArgs e)
        {
            this.Validate();
            животноеBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
        }

        private void butall_Click(object sender, EventArgs e)
        {
            // Убираем фильтр, чтобы показать все данные
            животноеBindingSource.RemoveFilter();

            // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
            обновлены
            this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);
        }

        private void butfind_Click(object sender, EventArgs e)
        {

```

```

        // Применяем фильтр по идентификационному номеру
        животноеBindingSource.Filter = "ФИО LIKE %" + textBox5.Text + "%";
    }

    private void butexit_Click(object sender, EventArgs e)
    {
        Form form1 = new Form1();
        form1.Show();
    }

    private void button2_Click(object sender, EventArgs e)
    {
        this.Validate();
        клиентBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }

    private void button3_Click(object sender, EventArgs e)
    {
        // Убираем фильтр, чтобы показать все данные
        клиентBindingSource.RemoveFilter();

        // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
        обновлены this.клиентTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Клиент);
    }

    private void button5_Click(object sender, EventArgs e)
    {
        клиентBindingSource.Filter = "ФИО Like " + textBox3.Text + "%";
    }

    private void bAdd_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из текстовых полей формы
            int Idanimal = (int)идентификационный_номерNumericUpDown1.Value; //
            Идентификационный номер животного
            int Iddoctor = (int)номер_врачаNumericUpDown1.Value; // Номер врача
            DateTime Datevisit = датаDateTimePicker1.Value; // Дата посещения
            string diagnosis = диагнозTextBox1.Text; // Диагноз
            int cost;

            // Проверка, что диагноз не пустой
            if (string.IsNullOrEmpty(diagnosis))
            {

```

```

        MessageBox.Show("Пожалуйста, введите диагноз.", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return; // Прерываем выполнение, если диагноз не введен
    }

    // Проверка на корректность итоговой стоимости
    if (!int.TryParse(итоговая_стоимостьTextBox1.Text, out cost))
    {
        MessageBox.Show("Некорректная итоговая стоимость. Пожалуйста,
        введите правильное число.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение, если стоимость не удалось
        преобразовать
    }

    // Проверка, что итоговая стоимость не меньше 0
    if (cost < 0)
    {
        MessageBox.Show("Итоговая стоимость не может быть меньше 0.",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение, если стоимость меньше 0
    }

    // Логируем полученные данные (для отладки)
    Console.WriteLine($"animalId: {Idanimal}, doctorId: {Iddoctor}, visitDate:
    {Datevisit.ToShortDateString()}, diagnosis: {diagnosis}, cost: {cost}");

    // Привязываем команду SQL
    sqlCommand3.Connection = sqlConnection3;
    sqlCommand3.CommandType = CommandType.StoredProcedure;
    sqlCommand3.CommandText = "AddVisit"; // Название вашей хранимой
    процедуры

    // Очистка старых параметров
    sqlCommand3.Parameters.Clear();

    // Добавление параметров (Без передачи Id посещения)
    sqlCommand3.Parameters.AddWithValue("@Идентификационный_номер",
    Idanimal);
    sqlCommand3.Parameters.AddWithValue("@Номер_врача", Iddoctor);
    sqlCommand3.Parameters.AddWithValue("@Дата", Datevisit);
    sqlCommand3.Parameters.AddWithValue("@Диагноз", diagnosis);
    sqlCommand3.Parameters.AddWithValue("@Итоговая_стоимость", cost);

    // Добавляем параметр для получения сообщения об ошибке
    SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
    SqlDbType.VarChar, 255);
    errorMessageParam.Direction = ParameterDirection.Output;
    sqlCommand3.Parameters.Add(errorMessageParam);

    // Открытие соединения с БД
    if (sqlConnection3.State != ConnectionState.Open)
        sqlConnection3.Open();

```



```

        // Выполнение команды, данные добавляются в базу
        sqlCommand3.ExecuteNonQuery(); // Используем ExecuteNonQuery для
        выполнения без возврата данных

        // Получаем возможное сообщение об ошибке
        string errorMessage = errorMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(errorMessage))
        {
            MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
            return;
        }

        // Получаем сгенерированный Id_посещения
        int generatedIdVisit = (int)sqlCommand3.ExecuteScalar();

        // Закрытие соединения
        sqlCommand3.Close();

        // После добавления — обновляем данные в таблице
        this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);

        // Сообщение об успешном добавлении
        MessageBox.Show($"Посещение успешно добавлено! Id посещения:
        {generatedIdVisit}", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (SqlException ex)
    {
        // Обработка ошибок, возникающих при выполнении процедуры
        MessageBox.Show($"Ошибка при добавлении посещения: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        // Обработка любых других ошибок
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void bSave_Click(object sender, EventArgs e)
{
    this.Validate();
    посещениеBindingSource1.EndEdit();
    this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
}

private void bAll_Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    посещениеBindingSource1.RemoveFilter();
}

```

```

        // Перегружаем данные, чтобы быть уверенным, что все строки корректно
обновлены
        this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);
    }

    private void bFind_Click(object sender, EventArgs e)
    {
        посещениеBindingSource1.Filter = "Id_посещения = " +
        numericUpDown2.Text + "";
    }

    private void buttadd_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из текстовых полей формы
            string FIO = textBox1.Text; // ФИО врача
            DateTime Birthday = dateTimePicker1.Value; // Дата рождения врача
            string Phone = телефонTextBox.Text; // Телефон врача
            string Speciality = должность_СпециальностьTextBox.Text; //
Должность/Специальность

            // Логируем полученные данные (для отладки)
            Console.WriteLine($"fullName: {FIO}, birthDate:
{Birthday.ToShortDateString()}, phone: {Phone}, position: {Speciality}");

            // Проверка на пустые поля
            if (string.IsNullOrEmpty(FIO) || string.IsNullOrEmpty(Phone) ||
string.IsNullOrEmpty(Speciality))
            {
                MessageBox.Show("Пожалуйста, заполните все обязательные поля.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return; // Прерываем выполнение метода, если есть пустые поля
            }

            // Проверка на корректность зарплаты
            int salary;
            if (!int.TryParse(зарплатаTextBox.Text, out salary))
            {
                MessageBox.Show("Некорректная зарплата. Пожалуйста, введите
правильное число.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Прерываем выполнение, если зарплата не корректна
            }

            // Проверка, что зарплата больше 30000
            if (salary <= 30000)
            {
                MessageBox.Show("Зарплата должна быть больше 30000.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Прерываем выполнение, если зарплата меньше 30000
            }
        }
    }

```

```

        // Привязываем команду SQL
        sqlCommand4.Connection = sqlConnection4;
        sqlCommand4.CommandType = CommandType.StoredProcedure;
        sqlCommand4.CommandText = "AddVetDoctor"; // Название вашей
хранимой процедуры

        // Очистка старых параметров
        sqlCommand4.Parameters.Clear();

        // Добавление параметров
        sqlCommand4.Parameters.AddWithValue("@ФИО", FIO);
        sqlCommand4.Parameters.AddWithValue("@Дата_рождения", Birthday);
        sqlCommand4.Parameters.AddWithValue("@Телефон", Phone);
        sqlCommand4.Parameters.AddWithValue("@Должность_Специальность",
Speciality);
        sqlCommand4.Parameters.AddWithValue("@Зарплата", salary);

        // Добавляем параметр для получения сообщения об ошибке
        SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
        errorMessageParam.Direction = ParameterDirection.Output;
        sqlCommand4.Parameters.Add(errorMessageParam);

        // Открытие соединения с БД
        if (sqlConnection4.State != ConnectionState.Open)
            sqlConnection4.Open();

        // Выполнение команды, данные добавляются в базу
        sqlCommand4.ExecuteNonQuery(); // Используем ExecuteNonQuery для
выполнения без возврата данных

        // Получаем возможное сообщение об ошибке
        string errorMessage = errorMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(errorMessage))
        {
            MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        // Закрытие соединения
        sqlConnection4.Close();

        // После добавления — обновляем данные в таблице

        this.ветеринарный_врачTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Ветеринарный_врач)
;

        // Сообщение об успешном добавлении
        MessageBox.Show("Ветеринарный врач успешно добавлен!", "Успех",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

```

```

        catch (SqlException ex)
        {
            // Обработка ошибок, возникающих при выполнении процедуры
            MessageBox.Show($"Ошибка при добавлении ветеринарного врача:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            // Обработка любых других ошибок
            MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void buttsave_Click(object sender, EventArgs e)
    {
        this.Validate();
        ветеринарный_врачBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }

    private void buttall_Click(object sender, EventArgs e)
    {
        // Убираем фильтр, чтобы показать все данные
        ветеринарный_врачBindingSource.RemoveFilter();

        // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
        обновлены

        this.ветеринарный_врачTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Ветеринарный_врач)
        ;
    }

    private void buttdelete_Click(object sender, EventArgs e)
    {
        try
        {
            // Удаляем текущего ветеринарного врача
            ветеринарный_врачBindingSource.RemoveCurrent();

            // Сохраняем изменения в базе данных
            this.Validate();
            ветеринарный_врачBindingSource.EndEdit();
            this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

            // Сообщение об успешном удалении
            MessageBox.Show("Ветеринарный врач успешно удален!", "Успех",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex)
        {
            // Обработка ошибок при удалении

```

```

        MessageBox.Show($"Ошибка при удалении ветеринарного врача:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void buttfind_Click(object sender, EventArgs e)
{
    ветеринарный_врачBindingSource.Filter = "ФИО Like " + textBox2.Text +
"%";
}

private void dataGridView1_CellContentClick(object sender,
DataGridViewCellEventArgs e)
{
}

private void tabPage2_Click(object sender, EventArgs e)
{
}

private void button1_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из текстовых полей формы
        string fullName = ФИОTextBox2.Text;
        string address = адресTextBox1.Text;
        string phone = телефонный_номерTextBox1.Text;
        string email = электронная_почтаTextBox1.Text;

        // Логируем полученные данные (для отладки)
        Console.WriteLine($"fullName: {fullName}, address: {address}, phone:
{phone}, email: {email}");

        // Проверка на пустые поля
        if (string.IsNullOrEmpty(fullName) || string.IsNullOrEmpty(address)
||
        string.IsNullOrEmpty(phone) || string.IsNullOrEmpty(email))
        {
            MessageBox.Show("Пожалуйста, заполните все поля.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return; // Прерываем выполнение метода, если есть пустые поля
        }

        // Привязываем команду SQL
        sqlCommand1.Connection = sqlConnection1;
        sqlCommand1.CommandType = CommandType.StoredProcedure;
        sqlCommand1.CommandText = "AddClient"; // Название вашей хранимой
процедуры

```

```

// Очистка старых параметров
sqlCommand1.Parameters.Clear();

// Добавление параметров без @Id_клиента
sqlCommand1.Parameters.AddWithValue("@ФИО", fullName);
sqlCommand1.Parameters.AddWithValue("@Адрес", address);
sqlCommand1.Parameters.AddWithValue("@Телефонный_номер", phone);
sqlCommand1.Parameters.AddWithValue("@Электронная_почта", email);

// Открытие соединения с БД
if (sqlConnection1.State != ConnectionState.Open)
    sqlConnection1.Open();

// Выполнение команды, данные добавляются в базу
sqlCommand1.ExecuteNonQuery(); // Используем ExecuteNonQuery для
выполнения без возврата данных

// Закрытие соединения
sqlConnection1.Close();

// После добавления — обновляем данные в таблице
this.клиентTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Клиент);

// Сообщение об успешном добавлении
MessageBox.Show("Клиент успешно добавлен!", "Успех",
MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch (SqlException ex)
{
    // Обработка ошибок, возникающих при выполнении процедуры
    MessageBox.Show($"Ошибка при добавлении клиента: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch (Exception ex)
{
    // Обработка любых других ошибок
    MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

private void button4_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из текстовых полей формы
        int idClient = (int)id_клиентаNumericUpDown.Value; // Идентификатор
клиента

        string Name = имяTextBox.Text; // Имя животного
        string View = видTextBox.Text; // Вид животного
        string Breed = породаTextBox.Text; // Порода животного

```

```

        DateTime Birthday = дата_рожденияDateTimePicker.Value; // Дата
рождения животного
        string Gender = полTextBox.Text; // Пол животного

        // Логируем полученные данные (для отладки)
        Console.WriteLine($"idClient: {idClient}, name: {Name}, species: {View},
breed: {Breed}, birthDate: {Birthday.ToShortDateString()}, gender: {Gender}");

        // Проверка на пустые поля
        if (string.IsNullOrEmpty(Name) || string.IsNullOrEmpty(View) ||
            string.IsNullOrEmpty(Breed) || string.IsNullOrEmpty(Gender))
        {
            MessageBox.Show("Пожалуйста, заполните все поля.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return; // Прерываем выполнение метода, если есть пустые поля
        }

        // Привязываем команду SQL
        sqlCommand2.Connection = sqlConnection2;
        sqlCommand2.CommandType = CommandType.StoredProcedure;
        sqlCommand2.CommandText = "AddAnimal"; // Название вашей хранимой
процедуры

        // Очистка старых параметров
        sqlCommand2.Parameters.Clear();

        // Добавление параметров (Без передачи идентификационного номера)
        sqlCommand2.Parameters.AddWithValue("@Id_клиента", idClient);
        sqlCommand2.Parameters.AddWithValue("@Имя", Name);
        sqlCommand2.Parameters.AddWithValue("@Вид", View);
        sqlCommand2.Parameters.AddWithValue("@Порода", Breed);
        sqlCommand2.Parameters.AddWithValue("@Дата_рождения", Birthday);
        sqlCommand2.Parameters.AddWithValue("@Пол", Gender);

        // Добавляем параметр для получения сообщения об ошибке
        SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
        errorMessageParam.Direction = ParameterDirection.Output;
        sqlCommand2.Parameters.Add(errorMessageParam);

        // Открытие соединения с БД
        if (sqlConnection2.State != ConnectionState.Open)
            sqlConnection2.Open();

        // Выполнение команды, данные добавляются в базу
        sqlCommand2.ExecuteNonQuery(); // Используем ExecuteNonQuery для
выполнения без возврата данных

        // Получаем возможное сообщение об ошибке
        string errorMessage = errorMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(errorMessage))
        {

```

```

        MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
        return;
    }

    // Закрытие соединения
    sqlConnection2.Close();

    // Перегружаем данные в таблице, чтобы увидеть изменения
    this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);

    // Сообщение об успешном добавлении
    MessageBox.Show("Животное успешно добавлено!", "Успех",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (SqlException ex)
    {
        // Обработка ошибок, возникающих при выполнении процедуры
        MessageBox.Show($"Ошибка при добавлении животного: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        // Обработка любых других ошибок
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button6_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем Id_посещения из NumericUpDown
        int visitId = (int)numericUpDown3.Value;

        // Вызываем метод для расчета итоговой стоимости
        int totalCost = CalculateTotalCost(visitId);

        // Отображаем итоговую стоимость в TextBox
        textBox4.Text = totalCost.ToString();
    }
    catch (FormatException)
    {
        MessageBox.Show("Пожалуйста, введите корректный ID посещения.",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Произошла ошибка: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```



```

    }
}

// Метод для расчета итоговой стоимости
private int CalculateTotalCost(int visitId)
{
    int totalCost = 0;

    // Строка подключения к базе данных
    string connectionString = "Data Source=LAPTOP-BQB68U1S;Initial
Catalog=ChausovaDaryaBD;Integrated Security=True;";

    // Создаем подключение к базе данных
    using (SqlConnection connection = new SqlConnection(connectionString))
    {
        try
        {
            // Открываем соединение
            connection.Open();

            // Создаем команду для вызова хранимой процедуры
            using (SqlCommand command = new
SqlCommand("РасчитатьИтоговуюСтоимость", connection))
            {
                command.CommandType = System.Data.CommandType.StoredProcedure;

                // Добавляем параметр Id_посещения
                command.Parameters.AddWithValue("@Id_посещения", visitId);

                // Выполняем команду и получаем результат
                SqlDataReader reader = command.ExecuteReader();

                // Если хранимая процедура вернула результат
                if (reader.Read())
                {
                    totalCost =
reader.GetInt32(reader.GetOrdinal("Итоговая_стоимость"));
                }
            }
        }
        catch (SqlException sqlEx)
        {
            MessageBox.Show($"Ошибка SQL: {sqlEx.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Произошла ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    return totalCost;
}

```

```

    }
    private void contextMenuStrip1_Opening(object sender, CancelEventArgs e)
    {
        }
    }
}

```

3. Форма «Ветеринарный врач»

```

public partial class Form4 : Form
{
    public Form4()
    {
        InitializeComponent();

        private void Form4_Load(object sender, EventArgs e)
        {
            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Ветеринарный_врач". При необходимости она может быть
            перемещена или удалена.

            this.ветеринарный_врачTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Ветеринарный_врач)
            ;

            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Клиент". При необходимости она может быть перемещена или
            удалена.

            this.клиентTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Клиент);

            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Посещение". При необходимости она может быть перемещена
            или удалена.

            this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);

            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.ПосещениеУслуга". При необходимости она может быть
            перемещена или удалена.

            this.посещениеУслугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ПосещениеУслуга);

            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Услуга". При необходимости она может быть перемещена или
            удалена.

            this.услугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Услуга);

            // TODO: данная строка кода позволяет загрузить данные в таблицу
            "chausovaDaryaBDDDataSet.Животное". При необходимости она может быть перемещена
            или удалена.

            this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);

        }

        private void butsave_Click_1(object sender, EventArgs e)
        {
            this.Validate();
            животноеBindingSource.EndEdit();

```

```

        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }

    private void butall_Click_1(object sender, EventArgs e)
    {
        // Убираем фильтр, чтобы показать все данные
        животноеBindingSource.RemoveFilter();

        // Перегружаем данные, чтобы быть уверенным, что все строки корректно
        обновлены
        this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);
    }

    private void butfind_Click(object sender, EventArgs e)
    {
        // Применяем фильтр по идентификационному номеру
        животноеBindingSource.Filter = "Идентификационный_номер=" +
        numericUpDown1.Text + "";
    }

    private void bAdd_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из текстовых полей формы
            int Idanimal = (int)numericUpDown3.Value; // Идентификационный номер
            животное
            int Iddoctor = (int)номер_врачаNumericUpDown.Value; // Номер врача
            DateTime Datevisit = датаDateTimePicker.Value; // Дата посещения
            string diagnosis = диагнозTextBox.Text; // Диагноз
            int cost;

            // Проверка, что диагноз не пустой
            if (string.IsNullOrEmpty(diagnosis))
            {
                MessageBox.Show("Пожалуйста, введите диагноз.", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return; // Прерываем выполнение, если диагноз не введен
            }

            // Проверка на корректность итоговой стоимости
            if (!int.TryParse(итоговая_стоимостьTextBox.Text, out cost))
            {
                MessageBox.Show("Некорректная итоговая стоимость. Пожалуйста,
                введите правильное число.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Прерываем выполнение, если стоимость не удалось
                преобразовать
            }

            // Проверка, что итоговая стоимость не меньше 0
            if (cost < 0)
            {

```

```

        MessageBox.Show("Итоговая стоимость не может быть меньше 0.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение, если стоимость меньше 0
    }

    // Логируем полученные данные (для отладки)
    Console.WriteLine($"animalId: {Idanimal}, doctorId: {Iddoctor}, visitDate:
{Datevisit.ToShortDateString()}, diagnosis: {diagnosis}, cost: {cost}");

    // Привязываем команду SQL
    sqlCommand2.Connection = sqlConnection2;
    sqlCommand2.CommandType = CommandType.StoredProcedure;
    sqlCommand2.CommandText = "AddVisit"; // Название вашей хранимой
процедуры

    // Очистка старых параметров
    sqlCommand2.Parameters.Clear();

    // Добавление параметров (Без передачи Id посещения)
    sqlCommand2.Parameters.AddWithValue("@Идентификационный_номер",
Idanimal);

    sqlCommand2.Parameters.AddWithValue("@Номер_врача", Iddoctor);
    sqlCommand2.Parameters.AddWithValue("@Дата", Datevisit);
    sqlCommand2.Parameters.AddWithValue("@Диагноз", diagnosis);
    sqlCommand2.Parameters.AddWithValue("@Итоговая_стоимость", cost);

    // Добавляем параметр для получения сообщения об ошибке
    SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
    errorMessageParam.Direction = ParameterDirection.Output;
    sqlCommand2.Parameters.Add(errorMessageParam);

    // Открытие соединения с БД
    if (sqlConnection2.State != ConnectionState.Open)
        sqlConnection2.Open();

    // Выполнение команды, данные добавляются в базу
    sqlCommand2.ExecuteNonQuery(); // Используем ExecuteNonQuery для
выполнения без возврата данных

    // Получаем возможное сообщение об ошибке
    string errorMessage = errorMessageParam.Value.ToString();
    if (!string.IsNullOrEmpty(errorMessage))
    {
        MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

    // Получаем сгенерированный Id посещения
    int generatedIdVisit = (int)sqlCommand2.ExecuteScalar();

```

```

        // Заккрытие соединения
        sqlConnection2.Close();

        // После добавления — обновляем данные в таблице
        this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);

        // Сообщение об успешном добавлении
        MessageBox.Show($"Посещение успешно добавлено! Id посещения:
{generatedIdVisit}", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (SqlException ex)
    {
        // Обработка ошибок, возникающих при выполнении процедуры
        MessageBox.Show($"Ошибка при добавлении посещения: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        // Обработка любых других ошибок
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void bSave_Click(object sender, EventArgs e)
{
    try
    {
        this.Validate();
        посещениеBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при сохранении: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void bAll_Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    посещениеBindingSource.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
    обновлены this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);

    // Уведомляем BindingSource, что данные изменились
    this.посещениеBindingSource.ResetBindings(true); // Обновляем все данные в
интерфейсе
}

```

```

private void bFind_Click(object sender, EventArgs e)
{
    посещениеBindingSource.Filter = "Id_посещения = '" + numericUpDown4.Text
+ "'";
}

private void buttAll_Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    услугаBindingSource.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
обновлены
    this.услугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Услуга);
}

private void button4_Click_1(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    посещениеУслугаBindingSource1.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
обновлены
    this.посещениеУслугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ПосещениеУслуга);
}

private void button2_Click_1(object sender, EventArgs e)
{
    посещениеУслугаBindingSource1.AddNew();
}

private void button3_Click_1(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из формы
        int номерУслуги = Convert.ToInt32(номер_услугиTextBox.Text); // Номер
услуги
        string названиеУслуги = название_услугиTextBox.Text; // Название услуги

        // 1. Проверка на существование услуги с таким Номером_услуги
        var услугаПоНомеру = this.chausovaDaryaBDDDataSet.Услуга
            .FirstOrDefault(u => u.Номер_услуги == номерУслуги);

        if (услугаПоНомеру == null)
        {
            // Если услуга с таким Номером_услуги не найдена

```

```

        MessageBox.Show("Услуга с таким номером не существует.", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение метода
    }

    // 2. Проверка на существование услуги с таким названием
    var услугаПоНазванию = this.chausovaDaryaBDDDataSet.Услуга
        .FirstOrDefault(u => u.Название_услуги == названиеУслуги);

    if (услугаПоНазванию == null)
    {
        // Если услуга с таким названием не найдена
        MessageBox.Show("Услуга с таким названием не существует.",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение метода
    }

    // 3. Проверка, что Номер_услуги соответствует Названию_услуги
    if (услугаПоНазванию.Номер_услуги != номерУслуги)
    {
        // Если номер услуги не совпадает с тем, что введено в форме
        MessageBox.Show("Номер услуги не соответствует названию услуги.",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение метода
    }

    // Если все проверки прошли, продолжаем выполнение и сохраняем
    изменения в таблице ПосещениеУслуга
    this.Validate();
    посещениеУслугаBindingSource1.EndEdit();
    this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

    // Если все прошло успешно
    MessageBox.Show("Данные успешно сохранены!", "Успех",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Если произошла ошибка, выводим сообщение с текстом ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button7_Click_1(object sender, EventArgs e)
{
    посещениеУслугаBindingSource1.Filter = "Id_посещения = '" +
    id_посещенияNumericUpDown2.Text + "'";
}

private void button1_Click_1(object sender, EventArgs e)

```

```

    {
        try
        {
            // Получаем данные из текстовых полей формы
            int idClient = (int)id_клиентаNumericUpDown.Value; // Идентификатор
клиента

            string Name = имяTextBox.Text; // Имя животного
            string View = видTextBox.Text; // Вид животного
            string Breed = породаTextBox.Text; // Порода животного
            DateTime Birthday = дата_рожденияDateTimePicker.Value; // Дата
рождения животного
            string Gender = полTextBox.Text; // Пол животного

            // Логируем полученные данные (для отладки)
            Console.WriteLine($"idClient: {idClient}, name: {Name}, species: {View},
breed: {Breed}, birthDate: {Birthday.ToShortDateString()}, gender: {Gender}");

            // Проверка на пустые поля
            if (string.IsNullOrEmpty(Name) || string.IsNullOrEmpty(View) ||
                string.IsNullOrEmpty(Breed) || string.IsNullOrEmpty(Gender))
            {
                MessageBox.Show("Пожалуйста, заполните все поля.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return; // Прерываем выполнение метода, если есть пустые поля
            }

            // Привязываем команду SQL
            sqlCommand1.Connection = sqlConnection1;
            sqlCommand1.CommandType = CommandType.StoredProcedure;
            sqlCommand1.CommandText = "AddAnimal"; // Название вашей хранимой
процедуры

            // Очистка старых параметров
            sqlCommand1.Parameters.Clear();

            // Добавление параметров (Без передачи идентификационного номера)
            sqlCommand1.Parameters.AddWithValue("@Id_клиента", idClient);
            sqlCommand1.Parameters.AddWithValue("@Имя", Name);
            sqlCommand1.Parameters.AddWithValue("@Вид", View);
            sqlCommand1.Parameters.AddWithValue("@Порода", Breed);
            sqlCommand1.Parameters.AddWithValue("@Дата_рождения", Birthday);
            sqlCommand1.Parameters.AddWithValue("@Пол", Gender);

            // Добавляем параметр для получения сообщения об ошибке
            SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
            errorMessageParam.Direction = ParameterDirection.Output;
            sqlCommand1.Parameters.Add(errorMessageParam);

            // Открытие соединения с БД
            if (sqlConnection1.State != ConnectionState.Open)
                sqlConnection1.Open();

```



```

        // Выполнение команды, данные добавляются в базу
        sqlCommand1.ExecuteNonQuery(); // Используем ExecuteNonQuery для
        выполнения без возврата данных

        // Получаем возможное сообщение об ошибке
        string errorMessage = errorMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(errorMessage))
        {
            MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
            return;
        }

        // Закрытие соединения
        sqlConnection1.Close();

        // После добавления — обновляем данные в таблице
        // Перезагружаем данные в таблице, чтобы увидеть изменения
        this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);

        // Сообщение об успешном добавлении
        MessageBox.Show("Животное успешно добавлено!", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (SqlException ex)
    {
        // Обработка ошибок, возникающих при выполнении процедуры
        MessageBox.Show($"Ошибка при добавлении животного: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        // Обработка любых других ошибок
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button5_Click(object sender, EventArgs e)
{
    клиентBindingSource.Filter = "ФИО Like '" + фИОTextBox.Text + "%'";
}
}
}

```

4. Форма «Медицинский статистик»

```

public partial class Form5 : Form
{
    public Form5()
    {

```

```

        InitializeComponent();
    }

    private void посещениеBindingNavigatorSaveItem_Click(object sender, EventArgs
e)
    {
        this.Validate();
        this.посещениеBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }

    private void Form5_Load(object sender, EventArgs e)
    {
        // TODO: данная строка кода позволяет загрузить данные в таблицу
"chausovaDaryaBDDDataSet.Животное". При необходимости она может быть перемещена
или удалена.
        this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
"chausovaDaryaBDDDataSet.Ветеринарный_врач". При необходимости она может быть
перемещена или удалена.
        this.ветеринарный_врачTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Ветеринарный_врач)
;
        // TODO: данная строка кода позволяет загрузить данные в таблицу
"chausovaDaryaBDDDataSet.Клиент". При необходимости она может быть перемещена или
удалена.
        // TODO: данная строка кода позволяет загрузить данные в таблицу
"chausovaDaryaBDDDataSet.ПосещениеУслуга". При необходимости она может быть
перемещена или удалена.
        this.посещениеУслугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ПосещениеУслуга);
        this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
"chausovaDaryaBDDDataSet.ПосещениеУслуга". При необходимости она может быть
перемещена или удалена.
        this.посещениеУслугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ПосещениеУслуга);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
"chausovaDaryaBDDDataSet.Услуга". При необходимости она может быть перемещена или
удалена.
        this.услугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Услуга);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
"chausovaDaryaBDDDataSet.Посещение". При необходимости она может быть перемещена
или удалена.
        this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);
    }

    private void bAdd__Click(object sender, EventArgs e)
    {
        try

```

```

{
    // Получаем данные из текстовых полей формы
    int Idanimal = (int)numericUpDown3.Value; // Идентификационный номер
животного

    int Iddoctor = (int)номер_врачаNumericUpDown1.Value; // Номер врача
    DateTime Datevisit = датаDateTimePicker1.Value; // Дата посещения
    string diagnosis = диагнозTextBox1.Text; // Диагноз
    int cost;

    // Проверка, что диагноз не пустой
    if (string.IsNullOrEmpty(diagnosis))
    {
        MessageBox.Show("Пожалуйста, введите диагноз.", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return; // Прерываем выполнение, если диагноз не введен
    }

    // Проверка на корректность итоговой стоимости
    if (!int.TryParse(итоговая_стоимостьTextBox1.Text, out cost))
    {
        MessageBox.Show("Некорректная итоговая стоимость. Пожалуйста,
        введите правильное число.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение, если стоимость не удалось
        преобразовать
    }

    // Проверка, что итоговая стоимость не меньше 0
    if (cost < 0)
    {
        MessageBox.Show("Итоговая стоимость не может быть меньше 0.",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение, если стоимость меньше 0
    }

    // Логируем полученные данные (для отладки)
    Console.WriteLine($"animalId: {Idanimal}, doctorId: {Iddoctor}, visitDate:
    {Datevisit.ToShortDateString()}, diagnosis: {diagnosis}, cost: {cost}");

    // Привязываем команду SQL
    sqlCommand1.Connection = sqlConnection1;
    sqlCommand1.CommandType = CommandType.StoredProcedure;
    sqlCommand1.CommandText = "AddVisit"; // Название вашей хранимой
процедуры

    // Очистка старых параметров
    sqlCommand1.Parameters.Clear();

    // Добавление параметров (Без передачи Id_посещения)
    sqlCommand1.Parameters.AddWithValue("@Идентификационный_номер",
    Idanimal);

    sqlCommand1.Parameters.AddWithValue("@Номер_врача", Iddoctor);
    sqlCommand1.Parameters.AddWithValue("@Дата", Datevisit);

```

```

sqlCommand1.Parameters.AddWithValue("@Диагноз", diagnosis);
sqlCommand1.Parameters.AddWithValue("@Итоговая_стоимость", cost);

// Добавляем параметр для получения сообщения об ошибке
SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
errorMessageParam.Direction = ParameterDirection.Output;
sqlCommand1.Parameters.Add(errorMessageParam);

// Открытие соединения с БД
if (sqlConnection1.State != ConnectionState.Open)
    sqlConnection1.Open();

// Выполнение команды, данные добавляются в базу
sqlCommand1.ExecuteNonQuery(); // Используем ExecuteNonQuery для
выполнения без возврата данных

// Получаем возможное сообщение об ошибке
string errorMessage = errorMessageParam.Value.ToString();
if (!string.IsNullOrEmpty(errorMessage))
{
    MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    return;
}

// Получаем сгенерированный Id_посещения
int generatedIdVisit = (int)sqlCommand1.ExecuteScalar();

// Закрытие соединения
sqlConnection1.Close();

// После добавления — обновляем данные в таблице
this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);

// Сообщение об успешном добавлении
MessageBox.Show($"Посещение успешно добавлено! Id посещения:
{generatedIdVisit}", "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch (SqlException ex)
{
    // Обработка ошибок, возникающих при выполнении процедуры
    MessageBox.Show($"Ошибка при добавлении посещения: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch (Exception ex)
{
    // Обработка любых других ошибок
    MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

```

```

private void bSave__Click(object sender, EventArgs e)
{
    try
    {
        this.Validate();
        посещениеBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при сохранении: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void bAll__Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    посещениеBindingSource.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
    обновлены this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);

    // Уведомляем BindingSource, что данные изменились
    this.посещениеBindingSource.ResetBindings(true); // Обновляем все данные в
    интерфейсе
}

private void bFind__Click(object sender, EventArgs e)
{
    посещениеBindingSource.Filter = "Id_посещения = '" + numericUpDown2.Text
    + "'";
}

private void button10_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из текстовых полей формы
        string названиеУслуги = textBox2.Text; // Название услуги
        int стоимость;

        // Проверка, что все обязательные поля заполнены
        if (string.IsNullOrEmpty(названиеУслуги))
        {
            MessageBox.Show("Пожалуйста, введите название услуги.", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return; // Прерываем выполнение, если название услуги пустое
        }
    }
}

```

```

// Проверка корректности стоимости
if (!int.TryParse(textBox1.Text, out стоимость))
{
    MessageBox.Show("Некорректная стоимость. Пожалуйста, введите
правильное число.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    return; // Прерываем выполнение, если стоимость не удалось
преобразовать
}

// Проверка, что стоимость не меньше 0
if (стоимость < 0)
{
    MessageBox.Show("Стоимость не может быть меньше 0.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    return; // Прерываем выполнение, если стоимость меньше 0
}

// Логируем полученные данные для отладки
Console.WriteLine($"Название услуги: {названиеУслуги}, Стоимость:
{стоимость}");

// Строка подключения к базе данных (замените на свои данные)
string connectionString = @"Data Source=LAPTOP-BQB68U1S;Initial
Catalog=ChausovaDaryaBD;Integrated Security=True;";

// Подключение к базе данных с использованием SqlConnection
using (SqlConnection connection = new SqlConnection(connectionString))
{
    // Открытие соединения с базой данных
    connection.Open();

    // Создание команды для выполнения запроса
    using (SqlCommand command = new SqlCommand("INSERT INTO
Услуга (Название_услуги, Стоимость) VALUES (@Название_услуги, @Стоимость)",
connection))
    {
        // Добавление параметров в команду
        command.Parameters.AddWithValue("@Название_услуги",
названиеУслуги);
        command.Parameters.AddWithValue("@Стоимость", стоимость);

        // Выполнение команды для добавления новой записи в таблицу
        command.ExecuteNonQuery();
    }

    // Заккрытие соединения
    connection.Close();
}

// Сообщение об успешном добавлении

```

```

        MessageBox.Show("Услуга успешно добавлена!", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);

        // После добавления — обновляем данные в таблице
        this.услугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Услуга);
    }
    catch (Exception ex)
    {
        // Обработка любых ошибок
        MessageBox.Show($"Ошибка при добавлении услуги: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button9_Click(object sender, EventArgs e)
{
    // Завершаем редактирование и сохраняем данные в базу данных
    this.Validate();
    this.услугаBindingSource.EndEdit();
    this.услугаTableAdapter.Update(this.chausovaDaryaBDDDataSet.Услуга);
    this.услугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Услуга);

}

private void button8_Click(object sender, EventArgs e)
{
    услугаBindingSource.Filter = "Название_услуги = '" + textBox3.Text + "'";
}

private void button1_Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    услугаBindingSource.RemoveFilter();

    // Перезагружаем все данные из базы данных
    this.услугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Услуга);
}

private void button13_Click(object sender, EventArgs e)
{
    посещениеУслугаBindingSource1.AddNew();
}

private void button12_Click(object sender, EventArgs e)
{

```

```

try
{
    // Получаем данные из формы
    int номерУслуги = Convert.ToInt32(номер_услугиTextBox.Text); // Номер
услуги

    string названиеУслуги = название_услугиTextBox.Text; // Название услуги

    // 1. Проверка на существование услуги с таким Номером_услуги
    var услугаПоНомеру = this.chausovaDaryaBDDDataSet.Услуга
        .FirstOrDefault(u => u.Номер_услуги == номерУслуги);

    if (услугаПоНомеру == null)
    {
        // Если услуга с таким Номером_услуги не найдена
        MessageBox.Show("Услуга с таким номером не существует.", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение метода
    }

    // 2. Проверка на существование услуги с таким названием
    var услугаПоНазванию = this.chausovaDaryaBDDDataSet.Услуга
        .FirstOrDefault(u => u.Название_услуги ==
названиеУслуги);

    if (услугаПоНазванию == null)
    {
        // Если услуга с таким названием не найдена
        MessageBox.Show("Услуга с таким названием не существует.",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение метода
    }

    // 3. Проверка, что Номер_услуги соответствует Названию_услуги
    if (услугаПоНазванию.Номер_услуги != номерУслуги)
    {
        // Если номер услуги не совпадает с тем, что введено в форме
        MessageBox.Show("Номер услуги не соответствует названию услуги.",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return; // Прерываем выполнение метода
    }

    // Если все проверки прошли, продолжаем выполнение и сохраняем
изменения в таблице ПосещениеУслуга
    this.Validate();

    // Явно вызываем методы, чтобы зафиксировать изменения
    посещениеУслугаBindingSource1.EndEdit(); // Завершаем редактирование

    this.посещениеУслугаTableAdapter.Update(this.chausovaDaryaBDDDataSet.ПосещениеУслуга)
; // Обновляем только изменения в таблице

    // Если все прошло успешно

```



```

        MessageBox.Show("Данные успешно сохранены!", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Если произошла ошибка, выводим сообщение с текстом ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button11_Click(object sender, EventArgs e)
{
    посещениеУслугаBindingSource1.Filter = "Id_посещения = " +
numericUpDown3.Value;
}

private void button4_Click_1(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    посещениеУслугаBindingSource1.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
обновлены
    this.посещениеУслугаTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ПосещениеУслуга);
}

private void button2_Click_1(object sender, EventArgs e)
{
    животноеBindingSource1.Filter = "Имя Like " + имяTextBox.Text + "%";
}

private void button3_Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    животноеBindingSource1.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
обновлены
    this.животноеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Животное);

    // Уведомляем BindingSource, что данные изменились
    this.посещениеBindingSource.ResetBindings(true); // Обновляем все данные в
интерфейсе
}
}
}

```

5. Форма «Менеджер по закупкам»

```

public partial class Form6 : Form
{
    public Form6()
    {
        InitializeComponent();
    }

    private void Form6_Load(object sender, EventArgs e)
    {
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.Медицинский_препарат". При необходимости она может быть
        перемещена или удалена.

        this.медицинский_препаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Медицинский_п
        репарат);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.ЗакупкаКорм". При необходимости она может быть
        перемещена или удалена.

        this.закупкаКормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ЗакупкаКорм);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.Посещение". При необходимости она может быть перемещена
        или удалена.
        this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.РасходКорм". При необходимости она может быть перемещена
        или удалена.
        this.расходКормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.РасходКорм);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.Расход". При необходимости она может быть перемещена или
        удалена.
        this.расходTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Расход);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.РасходПрепарат". При необходимости она может быть
        перемещена или удалена.

        this.расходПрепаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.РасходПрепарат);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.ЗакупкаКорм". При необходимости она может быть
        перемещена или удалена.
        this.закупкаTableAdapter1.Fill(this.chausovaDaryaBDDDataSet.Закупка);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.Закупка". При необходимости она может быть перемещена или
        удалена.

        this.закупкаПрепаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ЗакупкаПрепарат);
        // TODO: данная строка кода позволяет загрузить данные в таблицу
        "chausovaDaryaBDDDataSet.Корм". При необходимости она может быть перемещена или
        удалена.
        this.кормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Корм);
    }
}

```

```

    }

    private void tabPage4_Click(object sender, EventArgs e)
    {

    }

    private void tabPage2_Click(object sender, EventArgs e)
    {

    }

    private void Butadd_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из текстовых полей формы
            string название = названиеTextBox.Text.Trim(); // Название корма
            string производитель = производительTextBox1.Text.Trim(); //
Производитель
            string типКорма = тип_кормаTextBox.Text.Trim(); // Тип корма
            string единицаИзмерения = единица_измеренияTextBox.Text.Trim(); //
Единица измерения

            // Проверка на пустые поля
            if (string.IsNullOrEmpty(название))
            {
                MessageBox.Show("Пожалуйста, укажите название корма.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Прерываем выполнение метода, если поле пустое
            }

            if (string.IsNullOrEmpty(производитель))
            {
                MessageBox.Show("Пожалуйста, укажите производителя корма.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Прерываем выполнение метода, если поле пустое
            }

            if (string.IsNullOrEmpty(типКорма))
            {
                MessageBox.Show("Пожалуйста, укажите тип корма.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Прерываем выполнение метода, если поле пустое
            }

            if (string.IsNullOrEmpty(единицаИзмерения))
            {
                MessageBox.Show("Пожалуйста, укажите единицу измерения корма.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Прерываем выполнение метода, если поле пустое
            }
        }
    }

```

```

// Преобразуем строки в числа с помощью TryParse для проверки
количества
int количество;
if (!int.TryParse(количествоTextBox.Text, out количество) || количество <=
0)
{
    MessageBox.Show("Некорректное количество! Введите положительное
число.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    return; // Прерываем выполнение метода, если количество некорректно
}

// Проверка на цену
int цена;
if (!int.TryParse(ценаTextBox.Text, out цена) || цена <= 0)
{
    MessageBox.Show("Некорректная цена! Введите положительное число.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    return; // Прерываем выполнение метода, если цена некорректна
}

// Логируем полученные данные (для отладки)
Console.WriteLine($"Название: {название}, Производитель:
{производитель}, Тип: {типКорма}, Количество: {количество}, Единица измерения:
{единицаИзмерения}, Цена: {цена}");

// Привязываем команду SQL
sqlCommand1.Connection = sqlConnection1;
sqlCommand1.CommandType = CommandType.StoredProcedure;
sqlCommand1.CommandText = "AddFood"; // Название вашей хранимой
процедуры

// Очистка старых параметров
sqlCommand1.Parameters.Clear();

// Добавление параметров (Номер_корма не передаем, так как он
генерируется автоматически)
sqlCommand1.Parameters.AddWithValue("@Название", название);
sqlCommand1.Parameters.AddWithValue("@Производитель",
производитель);
sqlCommand1.Parameters.AddWithValue("@Тип_корма", типКорма);
sqlCommand1.Parameters.AddWithValue("@Количество", количество);
sqlCommand1.Parameters.AddWithValue("@Единица_измерения",
единицаИзмерения);
sqlCommand1.Parameters.AddWithValue("@Цена", цена);

// Добавляем параметры для получения сообщений
SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
errorMessageParam.Direction = ParameterDirection.Output;
sqlCommand1.Parameters.Add(errorMessageParam);

```

```

        SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255);
        successMessageParam.Direction = ParameterDirection.Output;
        sqlCommand1.Parameters.Add(successMessageParam);

        // Открытие соединения с БД
        if (sqlConnection1.State != ConnectionState.Open)
            sqlConnection1.Open();

        // Выполнение команды, данные добавляются в базу
        sqlCommand1.ExecuteNonQuery(); // Используем ExecuteNonQuery для
выполнения без возврата данных

        // Получаем сообщение об ошибке и успехе
        string errorMessage = errorMessageParam.Value.ToString();
        string successMessage = successMessageParam.Value.ToString();

        if (!string.IsNullOrEmpty(errorMessage))
        {
            // Если есть ошибка, показываем ошибку
            MessageBox.Show($"Ошибка: {errorMessage}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else if (!string.IsNullOrEmpty(successMessage))
        {
            // Если сообщение об успехе, показываем успех
            MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        // Закрытие соединения
        sqlConnection1.Close();

        // Перезагружаем данные в таблице, чтобы увидеть изменения
        this.кормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Корм);
    }
    catch (SqlException ex)
    {
        // Обработка ошибок, возникающих при выполнении процедуры
        MessageBox.Show($"Ошибка при добавлении корма: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        // Обработка любых других ошибок
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void Butsave_Click(object sender, EventArgs e)
{
    this.Validate();
}

```

```

        кормBindingSource.EndEdit();
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
    }

    private void Butall_Click(object sender, EventArgs e)
    {
        // Убираем фильтр, чтобы показать все данные
        кормBindingSource.RemoveFilter();

        // Перегружаем данные, чтобы быть уверенным, что все строки корректно
        обновлены this.кормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Корм);
    }

    private void button5_Click(object sender, EventArgs e)
    {
        кормBindingSource.RemoveCurrent();
    }

    private void Butfind_Click(object sender, EventArgs e)
    {
        кормBindingSource.Filter = "Название LIKE '%" + comboBox1.Text + "%";
    }

    private void button1_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из текстовых полей
            string название = названиеTextBox2.Text.Trim();
            string производитель = производительTextBox2.Text.Trim();
            string единицаИзмерения = единица_измеренияTextBox2.Text.Trim();

            // Проверка на пустые поля
            if (string.IsNullOrEmpty(название))
            {
                MessageBox.Show("Пожалуйста, укажите название препарата.",
                "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            if (string.IsNullOrEmpty(производитель))
            {
                MessageBox.Show("Пожалуйста, укажите производителя препарата.",
                "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            if (string.IsNullOrEmpty(единицаИзмерения))
            {

```

```

        MessageBox.Show("Пожалуйста, укажите единицу измерения
препарата.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // Преобразуем строки в числа с помощью TryParse для проверки
количества
    int количество;
    if (!int.TryParse(количествоTextBox6.Text, out количество) || количество <=
0)
    {
        MessageBox.Show("Некорректное количество! Введите положительное
число.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // Проверка на цену
    int цена;
    if (!int.TryParse(ценаTextBox2.Text, out цена) || цена <= 0)
    {
        MessageBox.Show("Некорректная цена! Введите положительное число.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // Привязываем команду SQL
    sqlCommand2.Connection = sqlConnection2;
    sqlCommand2.CommandType = CommandType.StoredProcedure;
    sqlCommand2.CommandText = "AddMedicalDrug";

    // Очистка старых параметров
    sqlCommand2.Parameters.Clear();

    // Добавление параметров
    sqlCommand2.Parameters.AddWithValue("@Название", название);
    sqlCommand2.Parameters.AddWithValue("@Производитель",
производитель);
    sqlCommand2.Parameters.AddWithValue("@Количество", количество);
    sqlCommand2.Parameters.AddWithValue("@Единица_измерения",
единицаИзмерения);
    sqlCommand2.Parameters.AddWithValue("@Цена", цена);

    // Добавляем параметры для получения сообщений
    SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
    errorMessageParam.Direction = ParameterDirection.Output;
    sqlCommand2.Parameters.Add(errorMessageParam);

    SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255);
    successMessageParam.Direction = ParameterDirection.Output;
    sqlCommand2.Parameters.Add(successMessageParam);

```

```

        // Открытие соединения с БД
        if (sqlConnection2.State != ConnectionState.Open)
            sqlConnection2.Open();

        // Выполнение команды
        sqlCommand2.ExecuteNonQuery();

        string errorMessage = errorMessageParam.Value.ToString();
        string successMessage = successMessageParam.Value.ToString();

        if (!string.IsNullOrEmpty(errorMessage))
        {
            MessageBox.Show($"Ошибка: {errorMessage}", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else if (!string.IsNullOrEmpty(successMessage))
        {
            MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
        }

        sqlConnection2.Close();

        this.медицинский_препаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Медицинский_п
            репарат);

    }
    catch (SqlException ex)
    {
        MessageBox.Show($"Ошибка при добавлении препарата: {ex.Message}",
            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button2_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование изменений в BindingSource
        this.Validate(); // Завершаем редактирование данных
        this.медицинский_препаратBindingSource.EndEdit(); // Применяем
        изменения из BindingSource

        // Сохраняем данные в базу данных

        this.медицинский_препаратTableAdapter.Update(this.chausovaDaryaBDDDataSet.Медицински
            й_препарат); // Обновление только нужной таблицы
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при обновлении препарата: {ex.Message}",
            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```



```

        // Перезагружаем данные в таблицу (если требуется)

this.медицинский_препаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Медицинский_п
репарат);

        // Сообщение об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Сообщение об ошибке
        MessageBox.Show($"Ошибка при сохранении: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
private void button4_Click(object sender, EventArgs e)
{
    медицинский_препаратBindingSource.RemoveCurrent();

}
private void button3_Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    медицинский_препаратBindingSource.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
    обновлены

this.медицинский_препаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Медицинский_п
репарат);

}

private void button6_Click(object sender, EventArgs e)
{
    // Применяем фильтр
    медицинский_препаратBindingSource.Filter = "Название = '" +
    comboBox2.Text + "'";

}

private void b_add_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из формы
        int idПосещения;
        if (!int.TryParse(id_посещенияNumericUpDown.Text, out idПосещения) ||
        idПосещения <= 0)
        {

```

```

        MessageBox.Show("Пожалуйста, введите корректное значение для поля
'Id_посещения'.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // Преобразование id_посещения в числовой формат и проверка
    if (idПосещения <= 0)
    {
        MessageBox.Show("Id_посещения не может быть меньше или равно 0.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // Привязываем команду SQL
    sqlCommandRashod.Connection = sqlConnectionRashod;
    sqlCommandRashod.CommandType = CommandType.StoredProcedure;
    sqlCommandRashod.CommandText = "AddRashod"; // Название вашей
хранимой процедуры

    // Очистка старых параметров
    sqlCommandRashod.Parameters.Clear();

    // Добавление параметров
    sqlCommandRashod.Parameters.AddWithValue("@Id_посещения",
idПосещения);

    // Добавляем параметры для получения сообщений (ошибка и успех)
    SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
    errorMessageParam.Direction = ParameterDirection.Output;
    sqlCommandRashod.Parameters.Add(errorMessageParam);

    SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255);
    successMessageParam.Direction = ParameterDirection.Output;
    sqlCommandRashod.Parameters.Add(successMessageParam);

    // Открытие соединения с БД
    if (sqlConnectionRashod.State != ConnectionState.Open)
        sqlConnectionRashod.Open();

    // Выполнение команды
    sqlCommandRashod.ExecuteNonQuery();

    // Получаем сообщения об ошибке или успехе
    string errorMessage = errorMessageParam.Value.ToString();
    if (!string.IsNullOrEmpty(errorMessage))
    {
        MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

```

```

        string successMessage = successMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(successMessage))
        {
            MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        // Закрытие соединения
        sqlConnectionRashod.Close();

        // Обновление данных в таблице (если нужно)
        this.pasходTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Расход);
    }
    catch (SqlException ex)
    {
        MessageBox.Show($"Ошибка при добавлении расхода: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
private void b_save_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование и валидируем данные
        this.Validate();
        this.pasходBindingSource.EndEdit();

        // Обновляем данные в базе данных
        this.pasходTableAdapter.Update(this.chausovaDaryaBDDDataSet.Расход);

        // Применяем изменения к базе
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Информировать пользователя об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Обработка ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void b_all_Click(object sender, EventArgs e)

```

```

    {
        try
        {
            // Убираем фильтр, чтобы показать все данные
            расходBindingSource.RemoveFilter();

            // Перезагружаем данные из базы данных
            this.расходTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Расход);
        }
        catch (Exception ex)
        {
            // Обработка ошибки
            MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void b_find_Click(object sender, EventArgs e)
    {
        расходBindingSource.Filter = "Id_посещения = '" + numericUpDown4.Text +
""";

    }

    private void button13_Click(object sender, EventArgs e)
    {
        // Убираем фильтр, чтобы показать все данные
        посещениеBindingSource.RemoveFilter();

        // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
обновлены
        this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);
    }

    private void bFind_Click(object sender, EventArgs e)
    {
        посещениеBindingSource.Filter = "Id_посещения = '" + numericUpDown2.Text
+ """;

    }

    private void button21_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из формы
            int idПосещения;
            if (!int.TryParse(id_посещенияNumericUpDown.Text, out idПосещения) ||
idПосещения <= 0)
            {
                MessageBox.Show("Пожалуйста, введите корректное значение для поля
'Id_посещения'.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
        }
    }

```

```

    }

    // Преобразование id_посещения в числовой формат и проверка
    if (idПосещения <= 0)
    {
        MessageBox.Show("Id_посещения не может быть меньше или равно 0.",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // Привязываем команду SQL
    sqlCommandRashod.Connection = sqlConnectionRashod;
    sqlCommandRashod.CommandType = CommandType.StoredProcedure;
    sqlCommandRashod.CommandText = "AddRashod"; // Название вашей
хранимой процедуры

    // Очистка старых параметров
    sqlCommandRashod.Parameters.Clear();

    // Добавление параметров
    sqlCommandRashod.Parameters.AddWithValue("@Id_посещения",
idПосещения);

    // Добавляем параметры для получения сообщений (ошибка и успех)
    SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
    errorMessageParam.Direction = ParameterDirection.Output;
    sqlCommandRashod.Parameters.Add(errorMessageParam);

    SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255);
    successMessageParam.Direction = ParameterDirection.Output;
    sqlCommandRashod.Parameters.Add(successMessageParam);

    // Открытие соединения с БД
    if (sqlConnectionRashod.State != ConnectionState.Open)
        sqlConnectionRashod.Open();

    // Выполнение команды
    sqlCommandRashod.ExecuteNonQuery();

    // Получаем сообщения об ошибке или успехе
    string errorMessage = errorMessageParam.Value.ToString();
    if (!string.IsNullOrEmpty(errorMessage))
    {
        MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

    string successMessage = successMessageParam.Value.ToString();
    if (!string.IsNullOrEmpty(successMessage))

```

```

        {
            MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        // Закрытие соединения
        sqlConnectionRashod.Close();

        // Обновление данных в таблице (если нужно)
        this.pasходTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Расход);
    }
    catch (SqlException ex)
    {
        MessageBox.Show($"Ошибка при добавлении расхода: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button20_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование и валидируем данные
        this.Validate();
        this.pasходBindingSource.EndEdit();

        // Обновляем данные в базе данных
        this.pasходTableAdapter.Update(this.chausovaDaryaBDDDataSet.Расход);

        // Применяем изменения к базе
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Информировать пользователя об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Обработка ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button19_Click(object sender, EventArgs e)
{

```

```

try
{
    // Убираем фильтр, чтобы показать все данные
    расходBindingSource.RemoveFilter();

    // Перезагружаем данные из базы данных
    this.расходTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Расход);
}
catch (Exception ex)
{
    // Обработка ошибки
    MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

private void button23_Click(object sender, EventArgs e)
{
    расходBindingSource.Filter = "Id_посещения = '" +
id_посещенияNumericUpDown1.Text + "'";
}

private void button18_Click(object sender, EventArgs e)
{
    посещениеBindingSource.Filter = "Id_посещения = '" + numericUpDown8.Text
+ "'";
}

private void button8_Click(object sender, EventArgs e)
{
    // Убираем фильтр, чтобы показать все данные
    посещениеBindingSource.RemoveFilter();

    // Перезагружаем данные, чтобы быть уверенным, что все строки корректно
обновлены
    this.посещениеTableAdapter.Fill(this.chausovaDaryaBDDDataSet.Посещение);
}

private void button22_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из формы
        int idРасхода = (int)id_расходаNumericUpDown2.Value;
        int номерПрепарата = (int)номер_препаратаNumericUpDown2.Value;

        // Используем TryParse для безопасного преобразования строки в число
        int количество;
        if (!int.TryParse(количествоTextBox4.Text, out количество))
        {

```

```

        MessageBox.Show("Введите корректное количество (целое число).",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }

    // Создаем объект SqlCommand для хранимой процедуры
    SqlCommand sqlCommandRashodPreperat = new
SqlCommand("AddRashodPreperat", sqlConnectionRashodPrep)
    {
        CommandType = CommandType.StoredProcedure
    };

    // Добавляем параметры
    sqlCommandRashodPreperat.Parameters.AddWithValue("@Id_расхода",
idРасхода);

    sqlCommandRashodPreperat.Parameters.AddWithValue("@Номер_препарата",
номерПрепарата);
    sqlCommandRashodPreperat.Parameters.AddWithValue("@Количество",
количество);

    SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255)
    {
        Direction = ParameterDirection.Output
    };
    sqlCommandRashodPreperat.Parameters.Add(errorMessageParam);

    SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255)
    {
        Direction = ParameterDirection.Output
    };
    sqlCommandRashodPreperat.Parameters.Add(successMessageParam);

    // Открытие соединения с БД
    if (sqlConnectionRashodPrep.State != ConnectionState.Open)
        sqlConnectionRashodPrep.Open();

    // Выполнение команды
    sqlCommandRashodPreperat.ExecuteNonQuery();

    // Получаем сообщения об ошибке или успехе
    string errorMessage = errorMessageParam.Value.ToString();
    if (!string.IsNullOrEmpty(errorMessage))
    {
        MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        sqlConnectionRashodPrep.Close();
        return;
    }

```



```

        string successMessage = successMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(successMessage))
        {
            MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
        }

        // После добавления — обновляем данные в таблице

this.расходПрепаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.РасходПрепарат);

        // Закрываем соединение с БД
        sqlConnectionRashodPrep.Close();
    }
    catch (SqlException ex)
    {
        MessageBox.Show($"Ошибка при добавлении расхода препарата:
        {ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
private void button25_Click(object sender, EventArgs e)
{
    try
    {
        // Прежде чем показывать все данные, убедимся, что сохранены все
        изменения
        if (this.Validate())
        {
            // Завершаем редактирование
            расходПрепаратBindingSource.EndEdit();

            // Обновляем базу данных
            this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
        }

        // Убираем фильтр, чтобы показать все данные
        расходПрепаратBindingSource.RemoveFilter();

        // Перегружаем данные в BindingSource

this.расходПрепаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.РасходПрепарат);
        расходПрепаратBindingSource.ResetBindings(false);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

    }
}

private void button24_Click(object sender, EventArgs e)
{
    расходПрепаратBindingSource.Filter = "Id_расхода = " + numericUpDown5.Text + """;
}

private void bt_find_Click(object sender, EventArgs e)
{
    расходКормBindingSource.Filter = "Id_расхода = " + numericUpDown1.Text + """;
}

private void bt_add_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из формы
        int idРасхода = (int)id_расходаNumericUpDown.Value;
        int номерКорма = (int)номер_кормаNumericUpDown2.Value;

        // Используем TryParse для безопасного преобразования строки в число
        int количество;
        if (!int.TryParse(количествоTextBox1.Text, out количество))
        {
            MessageBox.Show("Введите корректное количество (целое число).",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        // Создаем объект SqlCommand для хранимой процедуры
        SqlCommand sqlCommandRashodKorm = new
SqlCommand("AddRashodKorm", sqlConnectionRashodKorm)
        {
            CommandType = CommandType.StoredProcedure
        };

        // Добавляем параметры
        sqlCommandRashodKorm.Parameters.AddWithValue("@Id_расхода",
idРасхода);
        sqlCommandRashodKorm.Parameters.AddWithValue("@Номер_корма",
номерКорма);
        sqlCommandRashodKorm.Parameters.AddWithValue("@Количество",
количество);

        SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255)
        {
            Direction = ParameterDirection.Output
        }
    }
}

```

```

    };
    sqlCommandRashodKorm.Parameters.Add(errorMessageParam);

    SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255)
    {
        Direction = ParameterDirection.Output
    };
    sqlCommandRashodKorm.Parameters.Add(successMessageParam);

    // Открытие соединения с БД
    if (sqlConnectionRashodKorm.State != ConnectionState.Open)
        sqlConnectionRashodKorm.Open();

    // Выполнение команды
    sqlCommandRashodKorm.ExecuteNonQuery();

    // Получаем сообщения об ошибке или успехе
    string errorMessage = errorMessageParam.Value.ToString();
    if (!string.IsNullOrEmpty(errorMessage))
    {
        MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        sqlConnectionRashodKorm.Close();
        return;
    }

    string successMessage = successMessageParam.Value.ToString();
    if (!string.IsNullOrEmpty(successMessage))
    {
        MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }

    // После добавления — обновляем данные в таблице

    this.расходКормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.РасходКорм);

    // Закрываем соединение с БД
    sqlConnectionRashodKorm.Close();
}
catch (SqlException ex)
{
    MessageBox.Show($"Ошибка при добавлении расхода корма:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch (Exception ex)
{
    MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}
}

```

```

private void bt_all_Click(object sender, EventArgs e)
{
    try
    {
        // Прежде чем показывать все данные, убедимся, что сохранены все
изменения
        if (this.Validate())
        {
            // Завершаем редактирование
            расходКормBindingSource.EndEdit();

            // Обновляем базу данных
            this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
        }

        // Убираем фильтр, чтобы показать все данные
        расходКормBindingSource.RemoveFilter();

        // Перезагружаем данные в BindingSource, чтобы быть уверенным, что все
строки корректно обновлены

        this.расходКормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.РасходКорм);
        расходКормBindingSource.ResetBindings(false); // Сбрасываем привязки
после загрузки новых данных
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button10_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем данные из формы
        DateTime датаЗакупки = дата_закупкиDateTimePicker.Value;
        string поставщик = поставщикTextBox1.Text.Trim();
        DateTime датаПоставки = дата_поставкиDateTimePicker.Value;

        // Используем TryParse для безопасного преобразования строки в число
        int всего;
        if (!int.TryParse(итогоTextBox2.Text, out всего) || всего <= 0)
        {
            MessageBox.Show("Пожалуйста, введите корректное значение для поля
'Итого' (положительное число).", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return; // Прерываем выполнение метода, если поле некорректное
        }
    }
}

```

```

// Проверка на пустые поля
if (string.IsNullOrEmpty(поставщик))
{
    MessageBox.Show("Пожалуйста, укажите поставщика.", "Ошибка",
    MessageBoxButtons.OK, MessageBoxIcon.Error);
    return; // Прерываем выполнение метода, если поле пустое
}

// Проверка на корректность даты поставки
if (датаПоставки <= датаЗакупки)
{
    MessageBox.Show("Дата поставки должна быть позже даты закупки.",
    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    return; // Прерываем выполнение метода, если дата поставки
некорректна
}

// Привязываем команду SQL
sqlCommandZakupka.Connection = sqlConnectionZakupka;
sqlCommandZakupka.CommandType = CommandType.StoredProcedure;
sqlCommandZakupka.CommandText = "AddZakupka"; // Название вашей
хранимой процедуры

// Очистка старых параметров
sqlCommandZakupka.Parameters.Clear();

// Добавление параметров (Номер_закупки не передаем, так как он
генерируется автоматически)
sqlCommandZakupka.Parameters.AddWithValue("@Дата_закупки",
датаЗакупки);
sqlCommandZakupka.Parameters.AddWithValue("@Поставщик",
поставщик);
sqlCommandZakupka.Parameters.AddWithValue("@Дата_поставки",
датаПоставки);
sqlCommandZakupka.Parameters.AddWithValue("@Итого", всего);

// Добавляем параметры для получения сообщений (ошибка и успех)
SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255);
errorMessageParam.Direction = ParameterDirection.Output;
sqlCommandZakupka.Parameters.Add(errorMessageParam);

SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255);
successMessageParam.Direction = ParameterDirection.Output;
sqlCommandZakupka.Parameters.Add(successMessageParam);

// Открытие соединения с БД
if (sqlConnectionZakupka.State != ConnectionState.Open)
    sqlConnectionZakupka.Open();

// Выполнение команды

```

```

sqlCommandZakupka.ExecuteNonQuery();

// Получаем сообщения об ошибке или успехе
string errorMessage = errorMessageParam.Value.ToString();
if (!string.IsNullOrEmpty(errorMessage))
{
    MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    return;
}

string successMessage = successMessageParam.Value.ToString();
if (!string.IsNullOrEmpty(successMessage))
{
    MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
}

// Закрытие соединения
sqlConnectionZakupka.Close();

// После добавления — обновляем данные в таблице
this.закупкаTableAdapter1.Fill(this.chausovaDaryaBDDDataSet.Закупка);
}
catch (SqlException ex)
{
    MessageBox.Show($"Ошибка при добавлении закупки: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}
catch (Exception ex)
{
    MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
}
}

private void button9_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование
        this.Validate();
        закупкаBindingSource.EndEdit();

        // Сохраняем изменения в базе данных
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Проверка успешного сохранения
        MessageBox.Show("Данные успешно сохранены!", "Успех",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)

```

```

        {
            MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void button16_Click(object sender, EventArgs e)
    {
        try
        {
            // Убираем фильтр, чтобы показать все данные
            закупкаBindingSource.RemoveFilter();

            // Перезагружаем данные, чтобы быть уверенным, что все строки
корректно обновлены
            this.закупкаTableAdapter1.Fill(this.chausovaDaryaBDDDataSet.Закупка);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void button7_Click(object sender, EventArgs e)
    {
        закупкаBindingSource.Filter = "Дата_закупки = '" + dateTimePicker1.Text + "'";
    }

    private void button32_Click(object sender, EventArgs e)
    {
        try
        {
            // Убираем фильтр, чтобы показать все данные
            закупкаBindingSource.RemoveFilter();

            // Перезагружаем данные, чтобы быть уверенным, что все строки
корректно обновлены
            this.закупкаTableAdapter1.Fill(this.chausovaDaryaBDDDataSet.Закупка);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void button17_Click(object sender, EventArgs e)
    {
        закупкаBindingSource.Filter = "Дата_закупки = '" + dateTimePicker2.Text + "'";
    }

```

```

    }

    private void button31_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из формы
            int номерЗакупки = (int)номер_закупкиNumericUpDown3.Value;
            int номерПрепарата = (int)номер_препаратаNumericUpDown.Value;

            // Используем TryParse для безопасного преобразования строки в число
            int количество;
            if (!int.TryParse(количествоTextBox3.Text, out количество))
            {
                MessageBox.Show("Введите корректное количество (целое число).",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            // Создаем объект SqlCommand для хранимой процедуры
            SqlCommand sqlCommandZakupkaMed = new
SqlCommand("AddZakupkaDrug", sqlConnectionZakupkaMed)
            {
                CommandType = CommandType.StoredProcedure
            };

            // Добавляем параметры
            sqlCommandZakupkaMed.Parameters.AddWithValue("@Номер_закупки",
номерЗакупки);
            sqlCommandZakupkaMed.Parameters.AddWithValue("@Номер_препарата",
номерПрепарата);
            sqlCommandZakupkaMed.Parameters.AddWithValue("@Количество",
количество);

            SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255)
            {
                Direction = ParameterDirection.Output
            };
            sqlCommandZakupkaMed.Parameters.Add(errorMessageParam);

            SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255)
            {
                Direction = ParameterDirection.Output
            };
            sqlCommandZakupkaMed.Parameters.Add(successMessageParam);

            // Открытие соединения с БД
            if (sqlConnectionZakupkaMed.State != ConnectionState.Open)
                sqlConnectionZakupkaMed.Open();
        }
        catch { }
    }
}

```



```

        // Выполнение команды
        sqlCommandZakupkaMed.ExecuteNonQuery();

        // Получаем сообщения об ошибке или успехе
        string errorMessage = errorMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(errorMessage))
        {
            MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            sqlConnectionZakupkaMed.Close();
            return;
        }

        string successMessage = successMessageParam.Value.ToString();
        if (!string.IsNullOrEmpty(successMessage))
        {
            MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        // После добавления — обновляем данные в таблице

        this.закупкаПрепаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ЗакупкаПрепарат);
        sqlConnectionZakupkaMed.Close();
    }
    catch (SqlException ex)
    {
        MessageBox.Show($"Ошибка при добавлении препарата в закупку:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button30_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование
        this.Validate();
        закупкаПрепаратBindingSource.EndEdit();

        // Сохраняем изменения в базе данных
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Проверка успешного сохранения
        MessageBox.Show("Данные успешно сохранены!", "Успех",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

```

```

        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void button33_Click(object sender, EventArgs e)
    {
        try
        {
            // Убираем фильтр, чтобы показать все данные
            закупкаПрепаратBindingSource.RemoveFilter();

            // Перезагружаем данные с помощью TableAdapter

            this.закупкаПрепаратTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ЗакупкаПрепарат);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void button14_Click(object sender, EventArgs e)
    {
        try
        {
            // Получаем данные из формы
            int номерЗакупки = (int)номер_закупкиNumericUpDown1.Value;
            int номерКорма = (int)номер_кормаNumericUpDown1.Value;

            // Используем TryParse для безопасного преобразования строки в число
            int количество;
            if (!int.TryParse(количествоTextBox2.Text, out количество))
            {
                MessageBox.Show("Введите корректное количество (целое число).",
"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            // Создаем объект SqlCommand для хранимой процедуры
            SqlCommand sqlCommandZakupkaKorm = new
            SqlCommand("AddZakupkaFood", sqlConnectionZakupkaKorm)
            {
                CommandType = CommandType.StoredProcedure
            };

            // Добавляем параметры
            sqlCommandZakupkaKorm.Parameters.AddWithValue("@Номер_закупки",
номерЗакупки);

```

```

номерКорма);
sqlCommandZakupkaKorm.Parameters.AddWithValue("@Номер_корма",
количество);

SqlParameter errorMessageParam = new SqlParameter("@ErrorMessage",
SqlDbType.VarChar, 255)
{
    Direction = ParameterDirection.Output
};
sqlCommandZakupkaKorm.Parameters.Add(errorMessageParam);

SqlParameter successMessageParam = new SqlParameter("@SuccessMessage",
SqlDbType.VarChar, 255)
{
    Direction = ParameterDirection.Output
};
sqlCommandZakupkaKorm.Parameters.Add(successMessageParam);

// Открытие соединения с БД
if (sqlConnectionZakupkaKorm.State != ConnectionState.Open)
    sqlConnectionZakupkaKorm.Open();

// Выполнение команды
sqlCommandZakupkaKorm.ExecuteNonQuery();

// Получаем сообщения об ошибке или успехе
string errorMessage = errorMessageParam.Value.ToString();
if (!string.IsNullOrEmpty(errorMessage))
{
    MessageBox.Show(errorMessage, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    sqlConnectionZakupkaKorm.Close();
    return;
}

string successMessage = successMessageParam.Value.ToString();
if (!string.IsNullOrEmpty(successMessage))
{
    MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
}

// После добавления — обновляем данные в таблице

this.закупкаКормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ЗакупкаКорм);
sqlConnectionZakupkaKorm.Close();
}
catch (SqlException ex)
{
    MessageBox.Show($"Ошибка при добавлении корма в закупку:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

```

    }
    catch (Exception ex)
    {
        MessageBox.Show($"Неизвестная ошибка: {ex.Message}", "Ошибка",
        MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button29_Click(object sender, EventArgs e)
{
    закупкаПрепаратBindingSource.Filter = "Номер_закупки = '" +
    numericUpDown3.Text + "'";
}

private void button11_Click(object sender, EventArgs e)
{
    закупкаКормBindingSource.Filter = "Номер_закупки = '" +
    номер_закупкиNumericUpDown4.Text + "'";
}

private void button15_Click(object sender, EventArgs e)
{
    try
    {
        // Убираем фильтр, чтобы показать все данные
        закупкаКормBindingSource.RemoveFilter();

        // Перезагружаем данные с помощью TableAdapter
        this.закупкаКормTableAdapter.Fill(this.chausovaDaryaBDDDataSet.ЗакупкаКорм);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button9_Click_1(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование и валидируем данные
        this.Validate();
        this.закупкаBindingSource.EndEdit();

        // Обновляем данные в базе данных
        this.закупкаTableAdapter1.Update(this.chausovaDaryaBDDDataSet.Закупка);
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при обновлении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

```

```

        // Применяем изменения к базе
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Информировать пользователя об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Обработка ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button27_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование и валидируем данные
        this.Validate();
        this.закупкаПрепаратBindingSource.EndEdit();

        // Обновляем данные в базе данных
        this.закупкаПрепаратTableAdapter.Update(this.chausovaDaryaBDDDataSet.ЗакупкаПрепарат);

        // Применяем изменения к базе
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Информировать пользователя об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Обработка ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button28_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование и валидируем данные
        this.Validate();
        this.закупкаКормBindingSource.EndEdit();

        // Обновляем данные в базе данных

```

```

this.закупкаКормTableAdapter.Update(this.chausovaDaryaBDDDataSet.ЗакупкаКорм);

        // Применяем изменения к базе
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Информировать пользователя об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Обработка ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button12_Click(object sender, EventArgs e)
{
    // Удаляем текущего ветеринарного врача
    закупкаКормBindingSource.RemoveCurrent();

    // Сохраняем изменения в базе данных
    this.Validate();
    закупкаКормBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

}

private void button_del_Click(object sender, EventArgs e)
{
    // Удаляем текущего ветеринарного врача
    закупкаПрепаратBindingSource.RemoveCurrent();

    // Сохраняем изменения в базе данных
    this.Validate();
    закупкаПрепаратBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

}

private void button30_Click_1(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование и валидируем данные
        this.Validate();
        this.расходКормBindingSource.EndEdit();

        // Обновляем данные в базе данных

```

```

this.расходКормTableAdapter.Update(this.chausovaDaryaBDDDataSet.РасходКорм);

        // Применяем изменения к базе
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Информировать пользователя об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Обработка ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button34_Click(object sender, EventArgs e)
{
    try
    {
        // Завершаем редактирование и валидируем данные
        this.Validate();
        this.расходПрепаратBindingSource.EndEdit();

        // Обновляем данные в базе данных

        this.расходПрепаратTableAdapter.Update(this.chausovaDaryaBDDDataSet.РасходПрепарат);

        // Применяем изменения к базе
        this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);

        // Информировать пользователя об успешном сохранении
        MessageBox.Show("Данные успешно сохранены.", "Успех",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        // Обработка ошибки
        MessageBox.Show($"Ошибка при сохранении данных: {ex.Message}",
        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

private void button35_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем номер препарата из текстового поля

```

```

        int номерПрепарата =
        Convert.ToInt32(номер_препаратаNumericUpDown3.Text);

        // Подключаемся к базе данных
        using (SqlConnection connection = new SqlConnection("Data
Source=LAPTOP-BQB68U1S;Initial Catalog=ChausovaDaryaBD;Integrated Security=True"))
        {
            connection.Open();

            // Если необходимо, обновите данные в таблицах ЗакупкаПрепарат или
            РасходПрепарат
            // Здесь может быть код для обновления данных
            // Например: UpdateTableData(номерПрепарата);

            // Пересчитываем сумму препарата
            SqlCommand command1 = new
            SqlCommand("Рассчитать_Сумму_Препарата", connection);
            command1.CommandType = CommandType.StoredProcedure;

            // Добавляем параметр с явным указанием типа
            SqlParameter paramПрепарата1 = new
            SqlParameter("@Номер_препарата", SqlDbType.Int);
            paramПрепарата1.Value = номерПрепарата;
            command1.Parameters.Add(paramПрепарата1);

            // Читаем результат из первого запроса (сумма)
            SqlDataReader reader1 = command1.ExecuteReader();
            if (reader1.Read())
            {
                // Выводим результат в текстовое поле для суммы
                textBox8.Text = reader1["Сумма_Препарата"].ToString();
            }
            reader1.Close(); // Закрываем reader после первого запроса

            // Пересчитываем общее количество препарата
            SqlCommand command2 = new
            SqlCommand("Рассчитать_Общее_Количество_Препарата", connection);
            command2.CommandType = CommandType.StoredProcedure;

            // Добавляем параметр с явным указанием типа
            SqlParameter paramПрепарата2 = new
            SqlParameter("@Номер_препарата", SqlDbType.Int);
            paramПрепарата2.Value = номерПрепарата;
            command2.Parameters.Add(paramПрепарата2);

            // Читаем результат из второго запроса (общее количество)
            SqlDataReader reader2 = command2.ExecuteReader();
            if (reader2.Read())
            {
                // Выводим результат в текстовое поле для общего количества
                textBox6.Text = reader2["Общее_Количество_Препарата"].ToString();
            }
        }
    }
}

```



```

        reader2.Close(); // Закрываем второй reader
    }
}
catch (Exception ex)
{
    MessageBox.Show("Ошибка: " + ex.Message);
}
}

private void button36_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем номер корма из текстового поля
        int номерКорма = Convert.ToInt32(номер_кормаNumericUpDown.Text);

        // Подключаемся к базе данных
        using (SqlConnection connection = new SqlConnection("Data
Source=LAPTOP-BQB68UIS;Initial Catalog=ChausovaDaryaBD;Integrated Security=True"))
        {
            connection.Open();

            // Если необходимо, обновите данные в таблицах ЗакупкаКорм или
            // Здесь, например, может быть код для обновления данных
            // Example: UpdateTableData(номерКорма);

            // Пересчитываем сумму корма
            SqlCommand command1 = new
SqlCommand("Рассчитать_Сумму_Корма", connection);
            command1.CommandType = CommandType.StoredProcedure;

            // Добавляем параметр с явным указанием типа
            SqlParameter paramКорма1 = new SqlParameter("@Номер_корма",
SqlDbType.Int);
            paramКорма1.Value = номерКорма;
            command1.Parameters.Add(paramКорма1);

            // Читаем результат из первого запроса (сумма)
            SqlDataReader reader1 = command1.ExecuteReader();
            if (reader1.Read())
            {
                // Выводим результат в текстовое поле для суммы
                textBox7.Text = reader1["Сумма_Корма"].ToString();
            }
            reader1.Close(); // Закрываем первый reader

            // Пересчитываем общее количество корма
            SqlCommand command2 = new
SqlCommand("Рассчитать_Общее_Количество_Корма", connection);
            command2.CommandType = CommandType.StoredProcedure;

```

```

        // Добавляем параметр с явным указанием типа
        SqlParameter paramКорма2 = new SqlParameter("@Номер_корма",
SqlDbType.Int);
        paramКорма2.Value = номерКорма;
        command2.Parameters.Add(paramКорма2);

        // Читаем результат из второго запроса (общее количество)
        SqlDataReader reader2 = command2.ExecuteReader();
        if (reader2.Read())
        {
            // Выводим результат в текстовое поле для общего количества
            textBox1.Text = reader2["Общее_Количество_Корма"].ToString();
        }
        reader2.Close(); // Закрываем второй reader
    }
}
catch (Exception ex)
{
    MessageBox.Show("Ошибка: " + ex.Message);
}
}

private void button26_Click(object sender, EventArgs e)
{
    try
    {
        // Получаем номер закупки из текстового поля
        int номерЗакупки = Convert.ToInt32(номер_закупкиNumericUpDown.Text);

        // Подключаемся к базе данных
        using (SqlConnection connection = new SqlConnection("Data
Source=LAPTOP-BQB68UIS;Initial Catalog=ChausovaDaryaBD;Integrated Security=True"))
        {
            connection.Open();

            // Создаем команду для вызова хранимой процедуры
            SqlCommand command = new
SqlCommand("Рассчитать_Итоговую_Сумму_Закупки", connection);
            command.CommandType = CommandType.StoredProcedure;

            // Добавляем параметр с явным указанием типа
            SqlParameter paramЗакупки = new SqlParameter("@Номер_закупки",
SqlDbType.Int);
            paramЗакупки.Value = номерЗакупки;
            command.Parameters.Add(paramЗакупки);

            // Читаем результат
            SqlDataReader reader = command.ExecuteReader();
            if (reader.HasRows) // Проверка на наличие строк
            {
                reader.Read(); // Переходим к первой строке
                // Выводим результат в текстовое поле
            }
        }
    }
}

```

```

        textBox9.Text = reader["Итоговая_Сумма_Закупки"].ToString();
    }
    else
    {
        // Если нет данных, выводим сообщение
        MessageBox.Show("Не найдена информация по указанному номеру
закупки.");
    }
}
}
catch (Exception ex)
{
    MessageBox.Show("Ошибка: " + ex.Message);
}
}

private void button37_Click(object sender, EventArgs e)
{
    // Удаляем текущего ветеринарного врача
    расходКормBindingSource.RemoveCurrent();

    // Сохраняем изменения в базе данных
    this.Validate();
    расходКормBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
}

private void button38_Click(object sender, EventArgs e)
{
    // Удаляем текущего ветеринарного врача
    расходПрепаратBindingSource.RemoveCurrent();

    // Сохраняем изменения в базе данных
    this.Validate();
    расходПрепаратBindingSource.EndEdit();
    this.tableAdapterManager.UpdateAll(this.chausovaDaryaBDDDataSet);
}
}

```