МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных ветеринарной клиники с применением технологии миграция данных и объектов из одной СУБД в другую»

Выполнил студент Сосновец Мария Игоревна

(Ф.И.О.)

Руководитель работы ст. преп Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2024

Содержание

[Введение 4](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984086)

[1 Постановка задачи 5](#_Toc154050211)

[1.1 Обзор аналогичных решений 6](#_Toc154050212)

[1.2 Требования к проекту 6](#_Toc154050213)

[1.3 Вывод по разделу 6](#_Toc154050213)

[2 Проектирование базы данных 8](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984088)

[2.1 Определение вариантов использования 8](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984093)

[2.2 Диаграммы UML, взаимодействие всех компонентов 9](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984093)

[2.3 Вывод по разделу 10](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984093)

[3 Разработка объектов базы данных 11](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984092)

[3.1 Разработка таблиц базы данных 11](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984093)

[3.2 Разработка представлений 14](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984095)

[3.3 Разработка индексов 17](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984097)

[3.4 Разработка процедур 17](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984099)

[3.5 Разработка пользователей 18](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984099)

[3.5 Вывод по разделу 19](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984100)

[4 Описание процедур импорта и экспорта 20](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984101)

[4.1 Реализация процедур импорта и экспорта 20](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984101)

[4.2 Вывод по разделу 20](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984101)

[5 Тестирование производительности 22](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984102)

[5.1 Вывод по разделу 24](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984101)

[6 Описание технологии и ее применение в базе данных 25](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984103)

[6.1 Вывод по разделу 26](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984101)

7 Краткое описание приложения для демонстрации 27

[Заключение 28](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984105)

[Список используемых источников 29](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984106)

[Приложение А 30](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984107)

[Приложение Б 51](file:///D:\DOCUMENTS\google%20files\Пояснительная_БД_Трубач.docx#_Toc153984108)

Введение

В условиях стремительного развития информационных технологий управление данными становится ключевым аспектом успешного функционирования организаций, включая ветеринарные клиники. Данные о клиентах, сотрудниках, предоставляемых услугах и их стоимости требуют эффективного хранения, обработки и защиты. Одним из перспективных подходов к управлению данными является использование баз данных и технологий миграции данных между различными системами управления базами данных.

Актуальность разработки базы данных для ветеринарной клиники обусловлена необходимостью автоматизации управленческих процессов, снижения вероятности ошибок при работе с данными, а также обеспечением доступности информации для принятия обоснованных решений. Миграция данных между различными СУБД становится важной задачей в условиях обновления технологий, смены программного обеспечения или интеграции с внешними системами.

Основной целевой аудиторией данного продукта являются:

* администраторы ветеринарных клиник, заинтересованные в эффективном управлении данными;
* персонал клиники, включая ветеринаров и обслуживающий персонал, нуждающийся в удобных инструментах для работы;
* пользователи услуг клиники — клиенты, желающие получать актуальную информацию об услугах и специалистах.

Цель данной курсовой работы — разработать и реализовать базу данных для ветеринарной клиники с поддержкой миграции данных и объектов между различными СУБД.

Для того, чтобы реализовать курсовую работу необходимо решить следующие задачи:

* определить роли пользователей системы (администратор, пользователь, сотрудник);
* реализовать управление персоналом (добавление, удаление, изменение информации о сотрудниках);
* организовать управление услугами клиники (добавление, удаление, изменение информации о предоставляемых услугах);
* обеспечить функционал поиска и сортировки услуг для пользователей;
* реализовать поддержку миграции данных и объектов базы данных между различными СУБД.

Реализация данных задач позволит создать современную и удобную систему управления ветеринарной клиникой, обеспечивающую надежное хранение данных, удобство доступа к информации и возможность адаптации к изменяющимся технологическим требованиям.

1 Постановка задачи

* 1. **Обзор аналогичных решений**

Для создания эффективной системы управления базой данных парфюмерного магазина, важно провести анализ существующих аналогичных решений, чтобы выявить их сильные и слабые стороны, успешные подходы и потенциальные улучшения. Такой обзор позволяет определить ключевые функциональные требования и извлечь полезный опыт для разработки более удобной и функциональной базы данных. Анализ охватывает такие аспекты, как функциональность, пользовательский интерфейс, поддержка мультимедийных данных, управление запасами и интеграция с онлайн-платформами.

При создании базы данных для ветеринарной клиники необходимо изучить уже существующие решения, чтобы выявить ключевые сущности и процессы, а также определить функциональные требования. Рассмотрим три примера.

Первый пример — ветеринарная клиника «Главное Хвост». Основные элементы её базы данных включают животных (пациентов), врачей, предоставляемые услуги, записи на приём и истории болезней. Система позволяет регистрировать новых пациентов, записывать их на приём к специалистам, предоставлять медицинские услуги, вести медицинские карты и организовывать расписание врачей. Однако её недостатком можно считать ограниченные возможности для онлайн-записи.

Следующий пример — клиника «Питомец». Ключевыми сущностями этой базы данных являются владельцы животных (клиенты), сами питомцы, сотрудники, услуги и визиты. Основные функции включают регистрацию клиентов и их питомцев, запись на приём, предоставление услуг (например, вакцинацию и хирургические вмешательства), ведение учёта визитов и управление командой. Проблемой может быть отсутствие интеграции с внешними сервисами.

Третий пример — клиника «Sas Animal Service». В её базе данных присутствуют сущности такие, как врачи, пациенты, услуги, записи на приём и отзывы. К основным функциям относятся запись на приём через сайт, оказание различных ветеринарных услуг, управление базой данных врачей и их специализаций, а также сбор отзывов от клиентов. Возможными недостатками являются сложности с интеграцией системы напоминаний для клиентов.

Обзор вышеперечисленных известных аналогов решений, связанных с управлением базами данных ветеринарных клиник, позволяет проанализировать все преимущества и недостатки альтернативных возможностей реализации системы управления клиникой. Это также дает возможность сформулировать список требований к программному обеспечению, разрабатываемому в данном курсовом проекте. В процессе анализа были выделены ключевые аспекты, которые должны быть учтены при проектировании системы, такие как удобство интерфейса, безопасность данных, интеграция с другими системами (например, бухгалтерскими или медицинскими), а также возможность масштабирования для различных типов клиник.

Кроме того, было выявлено, что многие существующие решения либо имеют ограниченную функциональность, либо требуют значительных усилий для адаптации под специфические нужды отдельных клиник. В связи с этим, разработка новой системы должна обеспечивать гибкость в настройках, простоту в эксплуатации и высокий уровень поддержки пользователей.

* 1. **Требования к проекту**

Цель данного курсового проекта заключается в проектировании базы данных для ветеринарной клиники. Для этой цели будет использована реляционная модель данных. Проектирование должно быть выполнено с учетом общих требований к информации, хранящейся в базе данных. Функционально необходимо выполнить следующие задачи:

* определение ролей (администратор, пользователь, сотрудник);
* управление персоналом (добавление, удаление, изменение);
* управление услугами клиники (добавление, изменение, удаление);
* поиск услуг (пользователь);
* сортировка услуг;
* поддержка миграции данных и объектов клиники из одной СУБД в другую.

База данных должна быть реализована в СУБД PostgreSQL. Доступ к данным должен осуществляться только через соответствующие функции и хранимые процедуры. Количество объектов базы данных (таблиц, представлений, индексов, пользователей и т.д.) должно быть регламентировано задачей. Необходимо обеспечить импорт данных из файлов в формате JSON, а также экспорт данных в этот формат. Также потребуется провести тестирование производительности базы данных на таблице, содержащей не менее 100 000 строк, и в случае необходимости внести изменения в структуру базы данных. Важно провести анализ планов запросов к таблице для оптимизации работы базы данных.

* 1. **Вывод по разделу**

В данном разделе был проведен обзор существующих решений в области управления базами данных для ветеринарных клиник, что позволило выявить их основные достоинства и недостатки, а также определить функциональные требования к проектируемой системе. Анализ существующих примеров показал, что многие системы обладают ограниченной функциональностью, что может затруднять их использование в клиниках с более сложными потребностями. Также были выявлены проблемы, такие как отсутствие интеграции с внешними сервисами и ограничения в онлайн-записи, которые могут существенно снижать эффективность работы клиники.

Кроме того, в разделе сформулированы технические требования, включая использование реляционной модели данных в СУБД PostgreSQL, доступ к данным через хранимые процедуры, а также возможности импорта и экспорта данных в формат JSON. Важным аспектом является тестирование производительности на больших объемах данных, что позволит оптимизировать структуру базы данных и улучшить ее функциональность.

1. Проектирование модели базы данных

Проектирование модели базы данных является ключевым этапом в разработке информационной системы для ветеринарной клиники. На этом этапе определяется структура базы данных, которая будет использоваться для хранения и обработки информации, связанной с пациентами, услугами, персоналом и другими аспектами работы клиники. Важно, чтобы модель данных была гибкой, масштабируемой и легко адаптируемой под изменения, связанные с развитием клиники или внедрением новых функций. Этот процесс включает несколько важных этапов, начиная с определения требований пользователей, создания диаграммы вариантов использования, проектирования ролей пользователей, и заканчивая детализированным проектированием логической схемы базы данных, таблиц, хранимых процедур, представлений и индексов, что позволяет обеспечить надежную и эффективную работу системы.

* 1. **Определение вариантов использования**

Помимо функциональных требований, важно определить роли пользователей и сценарии их взаимодействия с системой. Сценарии использования (Use Cases) описывают, как пользователи будут работать с системой в рамках своих ролей. Это помогает задать, какие функции и данные должны быть доступны для каждой роли, а также продумать организацию навигации в системе. Сценарии использования часто представляются в виде диаграмм Use Case на языке UML, которые наглядно иллюстрируют взаимодействие пользователей с системой. На рисунке 2.1 представлена диаграмма вариантов использования.

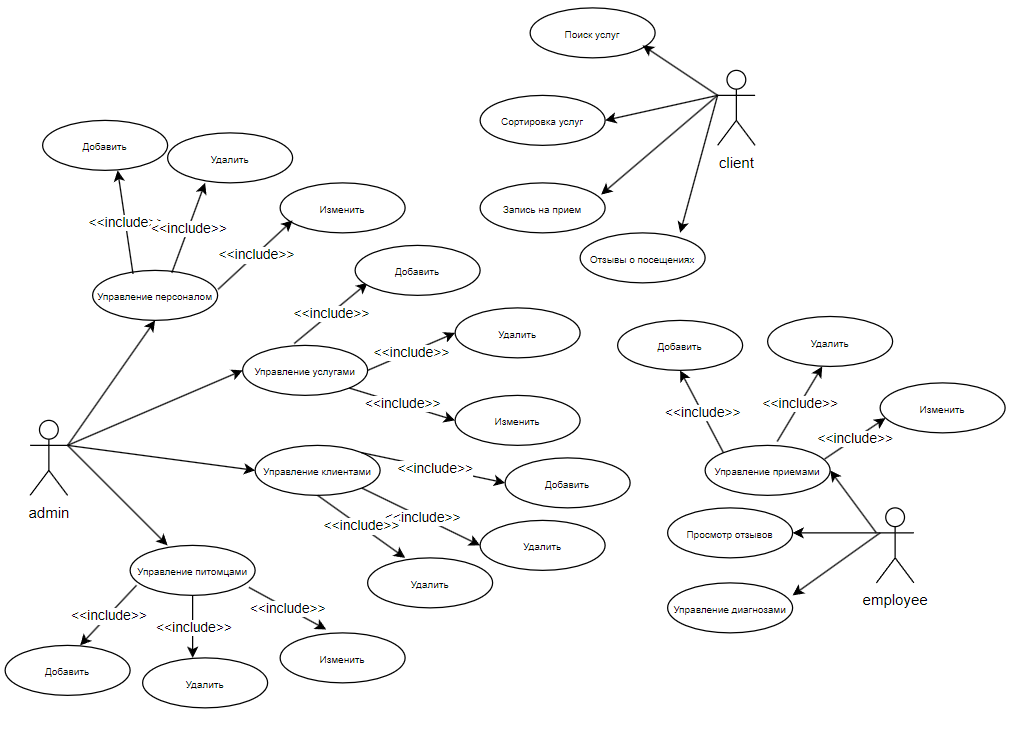


Рисунок 2.1 – UML диаграмма вариантов использования

Исходя из диаграммы Use Case на языке UML в системе выделяются три основные роли: клиент (client), администратор (admin) и сотрудник (employee), каждая из которых имеет свой набор функций и возможностей.

Клиент может искать и сортировать услуги, записываться на прием, а также оставлять отзывы о посещениях.

Роль администратора заключается в полном управлении системой. Администратор может добавлять, изменять и удалять данные о сотрудниках, клиентах, питомцах и услугах клиники, а также контролировать их актуальность.

Сотрудник работает с расписанием приемов, добавляя, изменяя или удаляя записи. Кроме того, сотрудник может управлять диагнозами пациентов и просматривать отзывы клиентов для улучшения качества обслуживания.

* 1. **Диаграммы UML, взаимодействие всех компонентов**

Схема таблиц базы данных (Database Table Diagram) представляет собой графическое изображение архитектуры базы данных и взаимосвязей между таблицами, которые находятся в этой базе. Схема базы данных иллюстрирована на рисунке 2.2.

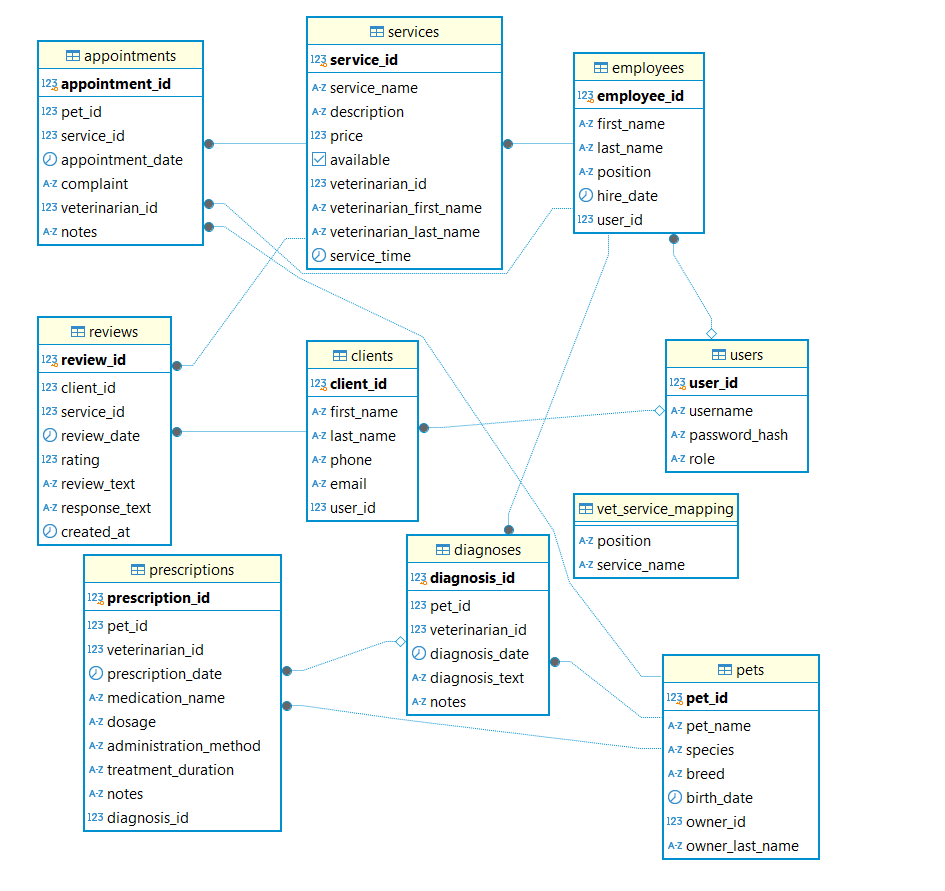


Рисунок 2.2 – Схема базы данных petclinic

Анализ структуры диаграммы базы данных показывает, что наиболее распространенным типом связи является "один ко многим" (1:N). Этот тип связи встречается чаще всего, поскольку он эффективно отражает отношения между основными сущностями базы данных ветеринарной клиники, такими как клиенты, питомцы, услуги, приемы и назначения.

Связь "один ко многим" означает, что одна запись в одной таблице может быть связана с несколькими записями в другой таблице. Однако каждая запись из второй таблицы связана только с одной записью из первой таблицы.

* 1. **Вывод по разделу**

Проектирование модели базы данных для ветеринарной клиники является критически важным этапом, который определяет структуру хранения и обработки информации. Это позволяет эффективно управлять данными о клиентах, питомцах, услугах и персонале, обеспечивая гибкость и масштабируемость системы. Важно, чтобы модель данных могла адаптироваться к изменениям в работе клиники, таким как расширение функционала или внедрение новых услуг. Использование диаграмм UML и анализа вариантов использования помогает точно определить роли пользователей и сценарии их взаимодействия с системой. Структура базы данных, включая связи между таблицами, должна быть спроектирована с учетом эффективного управления данными и обеспечения надежности работы системы.

1. Разработка объектов базы данных
   1. **Разработка таблиц базы данных**

Для работы с данными была создана база данных petclinic в системе управления базами данных PostgreSQL. В качестве инструментов для разработки использовались стандартные средства PostgreSQL для проектирования и реализации структуры базы данных.

База данных petclinic включает в себя таблицы, такие как users, clients, pet,services,appointments,diagnoses, reviews, prescriptions, vet\_service\_mapping, которые предназначены для хранения информации о клиентах, питомцах, услугах, сотрудниках клиники и т.д. Структура таблиц и их взаимосвязи были разработаны с учетом специфики работы клиники. Ниже представлена структура основных таблиц базы данных, их взаимосвязи и назначение.

В таблице 3.1 представлена структура таблицы USERS, которая хранит всех пользователей.

Таблица 3.1 – Структура таблицы USERS

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| user\_id | Уникальный идентификатор пользователя, первичный ключ | SERIAL |
| username | Имя пользователя, уникальное | VARCHAR(50) |
| password\_hash | |  | | --- | | Хэш пароля пользователя |  |  | | --- | |  | | TEXT |
| role | |  | | --- | | Роль пользователя (admin, employee, client) |  |  | | --- | |  | | VARCHAR(20) |

В таблице 3.2 представлена структура таблицы EMPLOYEES, которая хранит всех сотрудников.

Таблица 3.2 – Структура таблицы EMPLOYEES

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| employee\_id | Уникальный идентификатор избранного, первичный ключ | SERIAL |
| first\_name | Уникальный идентификатор пользователя, которому принадлежит избранное, вторичный ключ | |  | | --- | | VARCHAR(100) |  |  | | --- | |  | |
| last\_name | Фамилия сотрудника | |  | | --- | | VARCHAR(100) |  |  | | --- | |  | |
| position | Должность сотрудника | |  | | --- | | VARCHAR(100) |  |  | | --- | |  | |
| hire\_date | Дата приема на работу | DATE |
| user\_id | Идентификатор пользователя (связь с таблицей users), уникальный | INT |

Таблица CLIENTS хранит всех клиентов (таблица 3.3):

Таблица 3.3 – Структура таблицы CLIENTS

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| client\_id | Уникальный идентификатор корзины конкретного пользователя, вторичный ключ | SERIAL |
| first\_name | Уникальный идентификатор товара, добавленного в корзину, вторичный ключ | |  | | --- | | VARCHAR(100) |  |  | | --- | |  | |
| last\_name | Фамилия клиента | |  | | --- | | VARCHAR(100) |  |  | | --- | |  | |
| phone | |  | | --- | | Телефон клиента |  |  | | --- | |  | | |  | | --- | | VARCHAR(15) |  |  | | --- | |  | |
| email | |  | | --- | | Электронная почта клиента, уникальная |  |  | | --- | |  | | |  | | --- | | VARCHAR(100) |  |  | | --- | |  | |
| user\_id | Идентификатор пользователя (связь с таблицей users), уникальный | INT |

Таблица PETS представляет собой питомцев клиента (таблица 3.4):

Таблица 3.4 – Структура таблицы PETS

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| pet\_id | Уникальный идентификатор питомца, первичный ключ | SERIAL |
| pet\_name | Имя питомца | VARCHAR(100) |
| species | |  | | --- | |  |  |  | | --- | | Вид животного | | VARCHAR(50) |
| breed | Порода животного | VARCHAR(50) |
| birth\_date | |  | | --- | | Дата рождения питомца |  |  | | --- | |  | | DATE |
| owner\_id | Идентификатор владельца (связь с таблицей clients) | INT |

Таблица SERVICES хранит информацию об услугах (таблица 3.5):

Таблица 3.5 – Столбцы таблицы SERVICES

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| service\_id | Уникальный идентификатор товара, первичный ключ | SERIAL |
| service\_name | Название продукта | VARCHAR(100) |
| description | Идентификатор половой принадлежности товара, внешний ключ | TEXT |
| price | Цена товара | NUMERIC(10, 2) |
| |  | | --- | | available |  |  | | --- | |  | | Идентификатор бренда товара, внешний ключ | BOOLEAN |
| |  | | --- | | veterinarian\_id |  |  | | --- | |  | | Год выпуска товара | INT |
| veterinarian\_first\_name | Количество данного товара на складе | VARCHAR(100) |
| veterinarian\_last\_name | Фото товара | VARCHAR(100) |
| service\_time | Описание товара | TIMESTAMP |

В таблице 3.6 представлена структура таблицы APPOINTMENTS, содержащая информацию о записях на прием.

Таблица 3.6 – Столбцы таблицы APPOINTMENTS

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| appointment\_id | Уникальный идентификатор пользователя, первичный ключ | SERIAL |
| pet\_id | Идентификатор питомца (связь с таблицей pets) | INT |
| service\_id | Идентификатор услуги (связь с таблицей services) | INT |
| appointment\_date | Дата и время приема | TIMESTAMP |
| complaint | Жалобы клиента | TEXT |
| veterinarian\_id | Идентификатор ветеринара (связь с таблицей employees) | INT |
| notes | Заметки ветеринара | TEXT |

Таблица REVIEWS хранит отзывы на услуги(таблица 3.7):

Таблица 3.7 – Структура таблицы REVIEWS

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| review\_id | Уникальный идентификатор отзыва, первичный ключ | SERIAL |
| client\_id | Идентификатор клиента (связь с таблицей clients) | INT |
| service\_id | Идентификатор услуги (связь с таблицей services) | INT |
| review\_date | Дата отзыва | TIMESTAMP |
| rating | Оценка услуги (от 1 до 5) | INT |
| review\_text | Текст отзыва | TEXT |
| response\_text | Ответ на отзыв (если есть) | TEXT |
| created\_at | Дата и время создания отзыва | TIMESTAMP |

Таблица 3.8 содержит структуру таблицы PRESCRIPTION, которая хранит информацию о рецепте прописанным сотрудником.

Таблица 3.8 – Структура таблицы PRESCRIPTION

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| prescription\_id | Уникальный идентификатор рецепта, первичный ключ | SERIAL |
| pet\_id | Идентификатор питомца (связь с таблицей pets) | INT |
| veterinarian\_id | Идентификатор ветеринара (связь с таблицей employees) | INT |
| prescription\_date | Дата назначения рецепта | TIMESTAMP |
| medication\_name | Название лекарства | VARCHAR(150) |
| dosage | Дозировка | VARCHAR(100) |
| administration\_method | Метод применения | VARCHAR(100) |
| treatment\_duration | Длительность лечения | VARCHAR(50) |
| notes | Заметки | TEXT |

Окончание таблицы 3.

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| Quantity | Количество определенного товара в заказе | INT |
| Price | Цена а товар | DECIMAL(6, 2) |

Таблица DIAGNOSIS хранит бренды товаров, доступных в каталоге (таблица 3.9):

Таблица 3.9 – Структура таблицы DIAGNOSIS

|  |  |  |
| --- | --- | --- |
| Название столбца | Описание | Тип |
| diagnosis\_id | Уникальный идентификатор диагноза, первичный ключ | SERIAL |
| pet\_id | Идентификатор питомца (связь с таблицей pets) | INT |
| veterinarian\_id | Идентификатор ветеринара, который поставил диагноз | INT |
| diagnosis\_date | Дата постановки диагноза | TIMESTAMP |
| diagnosis\_text | Текст диагноза | TEXT |
| notes | Дополнительные заметки по диагнозу | TEXT |

Таблица DIAGNOSIS хранит бренды товаров, доступных в каталоге (таблица 3.9):

* 1. **Разработка представлений**

Представление в базе данных представляет собой виртуальную таблицу, которая создается в результате выполнения запроса к одной или нескольким таблицам. Оно позволяет объединить данные из разных таблиц в одну, не затрагивая при этом оригинальные таблицы и их структуру. Представления служат для упрощения доступа к данным, их управления, а также для повышения эффективности выполнения запросов. В рамках данного проекта были разработаны представления:

* appointment\_details, представление, которое выводит информацию о записях на приём в ветеринарной клинике, объединяя данные из нескольких таблиц;
* client\_diagnoses, представление, которое выводит информацию о диагнозах питомцев, принадлежащих текущему пользователю (клиенту);
* all\_diagnoses, представление, которое показывает все диагнозы питомцев, включая имя питомца, текст диагноза, фамилию и имя ветеринара, а также дату постановки диагноза;
* employee\_view, представление, которое выводит информацию о сотрудниках клиники: идентификатор, имя, фамилию, должность и дату трудоустройства;
* employees\_clients\_view, представление, которое выводит информацию о клиентах, включая их идентификатор, имя, фамилию, номер телефона и email;
* employees\_pets\_view, представление, которое одержит данные о питомцах, включая их имя, вид, породу, дату рождения и информацию о владельце (имя и фамилия);
* employee\_services\_view, представление, которое выводит информацию о сервисах (услугах), которые предоставляет конкретный ветеринарный специалист;
* employee\_reviews\_view, представление, которое выводит отзывы клиентов о услугах, предоставленных конкретным ветеринаром;
* employee\_prescriptions\_view, представление, которое показывает рецепты, выписанные ветеринаром текущего пользователя;
* client\_pets\_view, представление, которое выводит информацию о питомцах текущего пользователя (клиента). Включает идентификатор питомца, его имя, вид, породу, дату рождения и идентификатор владельца;
* client\_pets, представление, которое выводит данные о питомцах текущего пользователя, такие же, как и в предыдущем представлении, но без явного фильтра по имени пользователя;
* admin\_pets\_view, представление, которое выводит информацию о питомцах и их владельцах для администратора. Включает имя питомца, вид, породу, дату рождения и полное имя владельца;
* client\_reviews\_view, представление, которое выводит отзывы клиентов о предоставленных услугах. Включает информацию о рейтинге, тексте отзыва, фамилии и имени ветеринара, а также дату отзывах;
* all\_reviews\_view, представление, которое выводит все отзывы, включая информацию о клиенте, ветеринаре, услуге и тексте отзывах.
* prescriptions\_view, представление, которое выводит данные о рецептах, выписанных ветеринаром, включая имя питомца, фамилию и имя ветеринара, диагноз, медикаменты, дозировку и другие параметры;
* client\_prescriptions\_view, представление, которое выводит информацию о рецептах, выписанных для питомцев текущего пользователя. Включает данные о питомце, ветеринаре, диагнозе, медикаментах и другие параметры.

Реализация нескольких из представлений представлено в листинге 3.1. Остальные представления будут аналогичны, работая с другими данными.

|  |
| --- |
| -- Представление, которое выводит информацию о питомцах текущего клиента  CREATE OR REPLACE VIEW client\_pets\_view AS  SELECT  p.pet\_id,  p.pet\_name,  p.species,  p.breed,  p.birth\_date,  p.owner\_id -- Этот столбец будет использоваться для поиска питомца  FROM pets p  JOIN clients c ON p.owner\_id = c.client\_id  JOIN users u ON c.user\_id = u.user\_id  WHERE u.username = CURRENT\_USER; -- Фильтруем питомцев только для текущего пользователя  -- Представление, которое выводит информацию о питомцах и их владельцах для администратора  CREATE OR REPLACE VIEW admin\_pets\_view AS  SELECT  p.pet\_id,  p.pet\_name,  p.species,  p.breed,  p.birth\_date,  CONCAT(c.first\_name, ' ', c.last\_name) AS owner\_name  FROM pets p  JOIN clients c ON p.owner\_id = c.client\_id  JOIN users u ON c.user\_id = u.user\_id; |

Листинг 3.1 – Представления client\_pets\_view, admin\_pets\_view

Представление client\_pets\_view было создано для того, чтобы клиент мог просматривать информацию о своих питомцах. Оно фильтрует данные только по питомцам, принадлежащим текущему пользователю, используя CURRENT\_USER. Таким образом, каждый клиент видит только информацию о своих питомцах.

Представление admin\_pets\_view создано для администраторов, чтобы они могли видеть информацию о всех питомцах в системе вместе с именами их владельцев. Оно не фильтрует данные по пользователю, что позволяет администратору получить полную картину всех питомцев в базе данных, включая информацию о владельцах.

## Разработка индексов

Индекс в базе данных — это специальная структура, предназначенная для ускорения поиска и обработки данных. Когда в таблице содержится большое количество записей, последовательный поиск может быть медленным. Индекс создается на основе значений одного или нескольких столбцов и ускоряет поиск, указывая на соответствующие строки таблицы.

Для повышения производительности базы данных в рамках данного проекта были созданы три индекса для таблицы reviews. Эти индексы обеспечивают более быстрый доступ к данным, позволяя эффективно фильтровать и находить нужные отзывы по определенным критериям, таким как услуги, клиенты или рейтинг. Создание индексов представлено в листинге 3.2.

|  |
| --- |
| -- Поиск по рейтингу  CREATE INDEX idx\_reviews\_rating ON reviews (rating);  -- Поиск по дате создания  CREATE INDEX idx\_reviews\_review\_date ON reviews (review\_date DESC); |

Листинг 3.2 – Индексы для поиска товаров по критериям

Использование индексов в базе данных существенно ускоряет выполнение операций поиска, сортировки и фильтрации данных, особенно при работе с большими объемами информации.

## Разработка процедур

Согласно требованиям к курсовой работе, доступ и работа с данными БД осуществляется через процедуры. Хранимая процедура — это набор предопределённых SQL-операций, которые хранятся в системе и могут быть выполнены по запросу. Они позволяют выполнять сложные операции с данными, включая выборку, вставку, обновление и удаление записей.

В рамках курсовой работы было реализовано около 35 процедур. В листинге 3.3 представлен пример процедуры delete\_appointment.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE delete\_appointment(  p\_appointment\_id INT -- Идентификатор записи на прием  )  LANGUAGE plpgsql AS $$  BEGIN  -- Проверяем существование записи на прием  IF NOT EXISTS (  SELECT 1 FROM appointments WHERE appointment\_id = p\_appointment\_id  ) THEN  RAISE EXCEPTION 'Запись на прием с ID % не найдена.', p\_appointment\_id;  END IF;  -- Удаление записи на прием  DELETE FROM appointments  WHERE appointment\_id = p\_appointment\_id;  RAISE NOTICE 'Запись на прием с ID % успешно удалена.', p\_appointment\_id;  END;  $$; |

Листинг 3.3 – Процедура удаления записи о приеме

Данная процедура используется администратором для удаления записи о приеме. Она принимает входные параметры, соответствующие полям, которые содержит каждый товар.

Скрипт других процедур, которые связаны с администратором, приведен в [приложении](#_Приложение_В) А.

## Разработка пользователей

Пользователь базы данных — это физическое или юридическое лицо, которое имеет права доступа к базе данных и использует её возможности для получения необходимой информации. В разные моменты жизненного цикла базы данных (проектирование, разработка, эксплуатация, обновление и реорганизация) взаимодействуют различные категории пользователей.

В процессе разработки базы данных были определены три категории пользователей: администратор, клиент, сотрудник.

Для добавления администратора был создан отдельный проект Admin.

Создание пользователей представлено в листинге 3.4.

|  |
| --- |
| -- Администратор  CREATE USER admin\_user WITH PASSWORD 'admin';  GRANT admin\_role TO admin\_user;  -- Сотрудник  CREATE USER employee\_user WITH PASSWORD 'Employee12345';  GRANT employee\_role TO employee\_user;  -- Клиент  CREATE USER client\_user WITH PASSWORD 'Client12345';  GRANT client\_role TO client\_user; |

Листинг 3.4 – Создание пользователей

Далее для созданного пользователя базы данных были выданы необходимые привилегии необходимые для функционала клиента.

## Вывод по разделу

В данной главе были рассмотрены основные объекты базы данных, разработка которых необходима для обеспечения эффективной работы информационной системы. В процессе проектирования и реализации были созданы таблицы, представляющие собой основную структуру для хранения данных. Для упрощения работы с данными и улучшения производительности были разработаны представления, которые позволяют объединять данные из разных таблиц без изменения их структуры.

Кроме того, в целях ускорения поиска и повышения производительности запросов были созданы индексы для наиболее часто используемых столбцов, что обеспечило более быстрый доступ к данным. Для реализации сложных операций и обработки данных были разработаны функции и процедуры, автоматизирующие различные задачи. Также были созданы триггеры для автоматического выполнения операций при изменении данных в таблицах.

Важным этапом разработки является создание пользователей базы данных, что обеспечивает управление доступом и защиту данных. Были определены различные типы пользователей с различными правами доступа, что способствует организации безопасной и эффективной работы системы.

1. Описание процедур импорта и экспорта

В качестве формата импорта и экспорта был выбран JSON. Сам формат JSON является одним из самых универсальных на текущий момент. В рамках курсового проекта были реализованы процедуры экспорта и импорта таблицы отзывов.

## Реализация процедур импорта и экспорта

Для таблицы REVIEWS в базе данных petclinic доступны процедуры экспорта и импорта отзывов в формате JSON. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в JSON, была разработана процедура, формирующая строку данных товара. Скрипт по созданию процедуры export\_reviews\_to\_json представлен в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE FUNCTION export\_reviews\_to\_json(p\_export\_path text)  RETURNS VOID  LANGUAGE plpgsql  AS $$  DECLARE  json\_text text;  BEGIN  -- Генерация JSON из данных таблицы  SELECT json\_agg(row\_to\_json(reviews))::text INTO json\_text FROM reviews;  -- Запись JSON в файл  EXECUTE 'COPY (SELECT ' || quote\_literal(json\_text) || '::text) TO ''' || p\_export\_path || ''' WITH (FORMAT text)';  END;  $$; |

Листинг 4.1 – Процедура export\_reviews\_to\_json

Для реализации импорта отзывов из JSON в базу данных, была разработана процедура, import\_reviews\_from\_json. Скрипт по ее созданию представлен в приложении Б.

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах JSON, что делает управление базой данных более удобным и эффективным.

## Вывод по разделу

В данном разделе рассматриваются разработанные механизмы обмена данными в формате JSON для таблицы отзывов базы данных petclinic. Данный формат был выбран благодаря его универсальности, широкому применению и удобству обработки.

Были реализованы процедуры экспорта и импорта данных, что позволяет пользователям базы данных создавать резервные копии отзывов, а также перемещать их между серверами. Процедуры ориентированы на поддержку удобного управления данными, минимизацию ошибок при передаче и упрощение администрирования базы данных.

1. Тестирование производительности

Для проверки производительности БД необходимо заполнить её большим набором данным и, сделав несколько запросов, проанализировать результат.

Для проверки производительности была выбрана таблица с отзывами в которую будет помещено 100 000 записей. Код заполняющей процедуры представлен на листинге 5.1.

|  |
| --- |
| INSERT INTO reviews (client\_id, service\_id, review\_date, rating, review\_text, response\_text)  SELECT  (SELECT client\_id FROM clients ORDER BY RANDOM() LIMIT 1), -- случайный client\_id  (SELECT service\_id FROM services ORDER BY RANDOM() LIMIT 1), -- случайный service\_id  NOW(), -- текущая дата  FLOOR(RANDOM() \* 5 + 1), -- случайная оценка от 1 до 5  md5(RANDOM()::TEXT), -- случайный текст отзыва  CASE  WHEN RANDOM() > 0.5 THEN md5(RANDOM()::TEXT) ELSE NULL  END  FROM generate\_series(1, 100000); |

Листинг 5.1 – Добавление строк в таблицу REVIEWS

На рисунке 5.1 представлен результат выполнения листинга 5.1

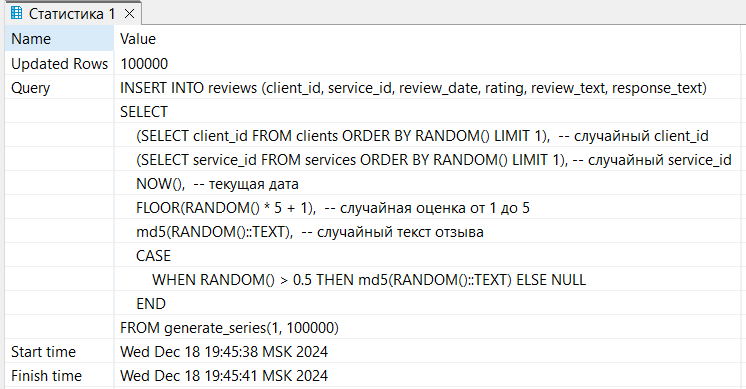


Рисунок 5.1 – Вставка строк в таблицу REVIEWS

Для первого теста использовался запрос, представленный в листинге 5.2, представляющий собой выборку по определенному бренду.

|  |
| --- |
| EXPLAIN ANALYZE  SELECT \*  FROM reviews  WHERE rating = 1  ORDER BY review\_date DESC; |

Листинг 5.2 – Запрос с условием к таблице REVIEWS

Запрос демонстрирует эффективное использование параллельного сканирования и сортировки, что способствует улучшению производительности. Однако использование только одного рабочего процесса может ограничивать эффективность, и увеличение их числа может привести к лучшим результатам при обработке больших объемов данных. Большое количество удалённых строк (80022) указывает на необходимость улучшения структуры данных или индексации для более эффективной выборки. Несмотря на то что общее время выполнения (42.225 мс) является приемлемым, его можно оптимизировать путем улучшения индексов и увеличения параллелизма. В целом, запрос работает эффективно, но дальнейшая оптимизация может повысить производительность и снизить затраты ресурсов.Результат запроса представлен на рисунке 5.2.

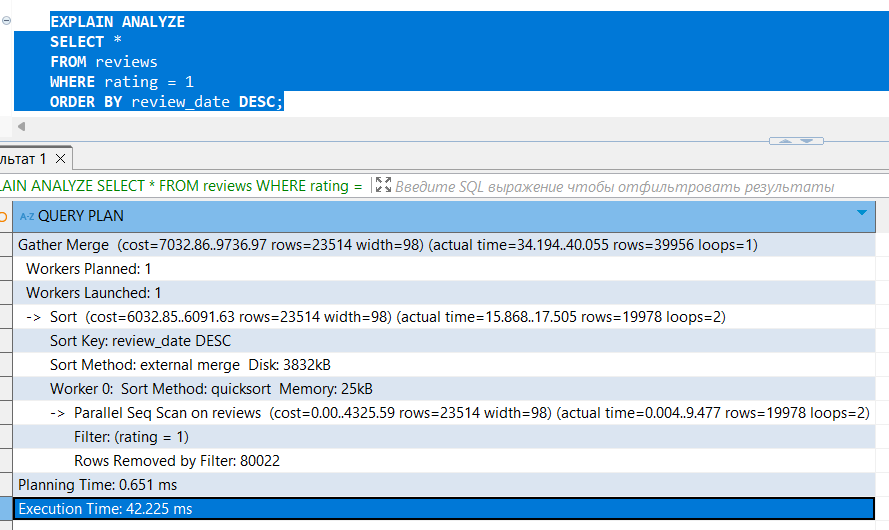


Рисунок 5.2 – Результат запроса

Для улучшения производительности был создан индекс покрытия idx\_reviews\_rating на поле rating, а также аналогичный индекс для поиска по дате. Создание индексов представлено в листинге 3.2.

Результаты выполнения запроса из листинга 5.2 после создания индексов представлены на рисунке 5.3.

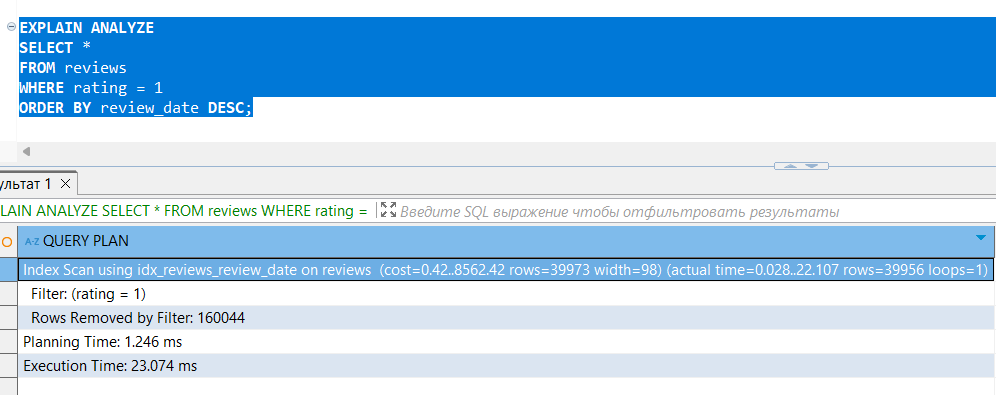


Рисунок 5.3 – Результат запроса с использованием индекса

Добавление индексов в базу данных может существенно повысить производительность запросов, ускоряя поиск и сортировку данных, а также снижая затраты на полное сканирование таблиц. Однако это также может негативно сказаться на производительности операций записи, так как индексы требуют обновления при изменении данных. Кроме того, индексы увеличивают объем используемого дискового пространства и могут усложнить управление базой данных. Важно тщательно оценить баланс между улучшением чтения данных и затратами на запись и хранение, чтобы оптимизировать общую производительность системы.

## Вывод по разделу

Таким образом, в этом разделе было проведено тестирование производительности базы данных на основе запросов, соответствующих вариантам использования. Тестирование показало, что база данных хорошо оптимизирована.

1. Описание технологии и ее применение в базе данных

Миграция данных — это процесс переноса данных из одной системы в другую. Этот процесс может включать перемещение данных между различными типами баз данных (например, из PostgreSQL в MySQL), между файлами и базами данных или между разными серверами и платформами.

Миграция данных используется в различных ситуациях. Например, при переходе на новую систему, когда организация решает заменить одну базу данных на другую из-за её производительности, удобства использования или других причин. Она необходима при объединении систем, когда две компании сливаются, и данные из их разных систем нужно переместить в единую платформу. В случае масштабирования системы при росте объёмов данных и повышении требований к производительности может потребоваться переход на более мощные решения. Миграция также важна для обеспечения совместимости, когда различные системы должны обмениваться данными в нужном формате. Кроме того, она используется для резервного копирования или архивирования, чтобы данные хранились долгое время в безопасных условиях.

Целями миграции являются сохранение целостности и актуальности данных, оптимизация производительности системы, а также поддержка новых технологий и программных решений.

В рамках разработки базы данных для ветеринарной клиники petclinic была применена технология миграции данных из одной СУБД в другую, что позволило эффективно переместить данные и объекты в другую СУБД.

Этапы миграции и инструменты миграции описаны далее.

Первым шагом является экспорт данных из PostgreSQL в формат CSV. Для этого используется команда \COPY, встроенная в PostgreSQL, которая сохраняет данные таблицы в файл на локальном диске. Пример команды показан в листинге 6.1:

|  |
| --- |
| \COPY users TO 'D:/masha/university/course/users.csv' DELIMITER ',' CSV HEADER; |

Листинг 6.1 – Экспорт данных из PostgreSQL в формат CSV

После экспорта данных необходимо создать соответствующие таблицы в MySQL. Для этого адаптируется SQL-код создания таблиц из PostgreSQL. Это может включать:

* преобразование типов данных;
* изменение синтаксиса индексов и ограничений.

Пример преобразования таблицы из PostgreSQL в MySql показан в листинге 6.2:

|  |
| --- |
| --PostgreSQL  CREATE TABLE IF NOT EXISTS users (  user\_id SERIAL PRIMARY KEY,  username VARCHAR(50) NOT NULL UNIQUE,  password\_hash TEXT NOT NULL,  role VARCHAR(20) NOT NULL CHECK (role IN ('admin', 'employee', 'client')) -- роли пользователей  );  --MySql  CREATE TABLE IF NOT EXISTS users (  user\_id INT AUTO\_INCREMENT PRIMARY KEY,  username VARCHAR(50) NOT NULL UNIQUE,  password\_hash TEXT NOT NULL,  role ENUM('admin', 'employee', 'client') NOT NULL -- Роли пользователей  ); |

Листинг 6.2 – Преобразование таблицы из PostgreSQL в MySql

После создания таблиц выполняется загрузка данных из ранее созданных CSV-файлов. Используется команда LOAD DATA LOCAL INFILE, которая загружает данные непосредственно в таблицы MySQL. Пример команды показан в листинге 6.3:

|  |
| --- |
| LOAD DATA LOCAL INFILE 'D:/masha/university/course/users.csv'  INTO TABLE users  FIELDS TERMINATED BY ','  ENCLOSED BY '"'  LINES TERMINATED BY '\n'  IGNORE 1 ROWS; |

Листинг 6.3 – Загрузка данных из CSV-файлов

После завершения процесса импорта данных проводится проверка их корректности и целостности.

## Вывод по разделу

Миграция данных из PostgreSQL в MySQL включает несколько этапов: экспорт данных, создание таблиц, загрузка информации и проверка корректности переноса. Успешная реализация требует учёта различий в структурах баз данных и синтаксисе SQL. Итогом является сохранение целостности данных и их корректное функционирование в новой системе.

1. Краткое описание приложения для демонстрации

База данных ветеринарной клиники предназначена для управления работой клиники, включая учет клиентов, сотрудников, животных, услуг, записей на прием, диагнозов, отзывов и рецептов. Система поддерживает работу с разными ролями пользователей: администратор, сотрудник и клиент, каждый из которых имеет доступ к определенным функциям.

Клиенты могут регистрироваться в системе, добавлять своих питомцев и записываться на приемы. Для этого клиент создает учетную запись с логином, паролем, контактными данными и основной информацией о питомце, такой как его имя, вид, порода и дата рождения. Запись на прием включает выбор услуги, указание времени и описания жалобы. После приема клиент может оставить отзыв о предоставленных услугах.

Сотрудники имеют доступ к информации о питомцах, назначенных услугах и диагнозах. Врачи могут добавлять, изменять и удалять услуги, связанные с их должностями, а также вести записи о диагнозах и назначенных рецептах. Для каждого диагноза сотрудник указывает текст диагноза, дату постановки и дополнительные примечания. Рецепты включают название лекарства, дозировку, способ применения и продолжительность лечения.

Администраторы управляют всеми аспектами базы данных. Они могут регистрировать сотрудников, назначать им должности, обновлять контактные данные клиентов и сотрудников, а также управлять услугами, отзывами, диагнозами и рецептами. Администраторы также могут просматривать действия всех пользователей и анализировать статистику системы.

Экспорт и импорт данных реализованы через работу с файлами формата JSON. Данные экспортируются в файл, который можно сохранить и использовать для восстановления базы данных. Для импорта данных файл загружается в базу данных, и его содержимое автоматически вставляется в соответствующие таблицы.

Поиск и сортировка предоставляют удобные инструменты для работы с услугами. Пользователь может найти услугу по ключевому слову или отсортировать список услуг по цене и фамилии врача. Это помогает быстро находить нужную информацию и упрощает управление записями.

Таким образом, база данных обеспечивает полное управление работой ветеринарной клиники, поддерживает многопользовательскую работу, предоставляет гибкие возможности поиска, сортировки, экспорта и импорта данных, а также обеспечивает контроль за действиями всех пользователей системы.

Заключение

База данных является важнейшим компонентом для эффективного управления информацией в современной ветеринарной клинике. В данной работе была поставлена задача разработки базы данных ветеринарной клиники с использованием СУБД PostgreSQL, обеспечивающей надежное хранение, обработку и управление данными.

В процессе разработки были созданы различные объекты базы данных, включая таблицы, представления, хранимые процедуры и функции, обеспечивающие структурированное хранение данных о клиентах, питомцах, услугах, приемах, диагнозах, рецептах и отзывах. Для поддержки безопасности системы были разработаны роли с разграничением доступа, что обеспечивает защиту данных и контроль над действиями пользователей.

Особое внимание было уделено реализации функций импорта и экспорта данных в формате JSON, что позволяет легко сохранять и восстанавливать данные. Экспорт данных осуществляется в JSON-файл, а импорт позволяет автоматически загружать данные обратно в базу. Это обеспечивает удобство работы с большими объемами информации.

База данных была протестирована на больших объемах данных, включая таблицы, содержащие более 100 000 записей. Проведенные тесты подтвердили ее высокую производительность и устойчивость при выполнении сложных запросов. Были проанализированы планы выполнения запросов, а структура базы данных оптимизирована для повышения скорости обработки данных.

Дополнительно была реализована система логирования действий пользователей, что позволяет отслеживать активность клиентов и сотрудников. Система также поддерживает сортировку и поиск услуг, обеспечивая быстрый доступ к нужной информации.

Таким образом, разработанная база данных ветеринарной клиники на основе СУБД PostgreSQL полностью соответствует поставленным требованиям. Она обеспечивает хранение, управление и защиту данных, поддерживает импорт и экспорт информации, а также предоставляет удобный интерфейс для работы пользователей различных ролей.

Список используемых источников

1. PostgreSQL Сайт о программировании [Электронный ресурс] / Режим доступа: https://postgrespro.ru/docs/postgresql.com – Дата доступа: 18.09.2024.
2. glavnoehvost.by/ [Электронный ресурс] / Режим https://www. glavnoehvost.by/ – Дата доступа: 21.09.2024.
3. petomec.by [Электронный ресурс] / Режим https://petomec.by/?ysclid=m4u61jux4o178960339– Дата доступа: 21.09.2024.
4. vetsas.by [Электронный ресурс] / Режим <https://vetsas.by/?ysclid=m4u61ltb4z170367614>/ – Дата доступа: 21.09.2024
5. Postgresqltutorial.com [Электронный ресурс] / Режим https://www.postgresqltutorial.com/ – Дата доступа: 21.09.2024.
6. Stackoverflow.com [Электронный https://stackoverflow.com – Дата доступа: 22.10.2023. ресурс] / Режим доступа: доступа: 30.09.2024
7. Миграция PostgreSQL. https://learn.microsoft.com/en-us/sql/integration services/import-export-data/connect-to-a-postgresql-data-source-sql-server-import-and export-wizard?view=sql-server-ver16 – Дата доступа: 02.12.2024.

Приложение А

Процедуры добавления, изменения и удаления записи на прием для админа

CREATE OR REPLACE PROCEDURE add\_appointment(

p\_pet\_name VARCHAR, p\_service\_name VARCHAR, p\_appointment\_date TIMESTAMP,

p\_veterinarian\_last\_name VARCHAR, p\_complaint TEXT, p\_notes TEXT DEFAULT NULL

)

LANGUAGE plpgsql AS $$

DECLARE

v\_pet\_id INT; v\_service\_id INT; v\_vet\_id INT;

v\_vet\_position VARCHAR; v\_client\_id INT;

v\_client\_first\_name VARCHAR; v\_client\_last\_name VARCHAR;

BEGIN

SELECT pet\_id, owner\_id INTO v\_pet\_id, v\_client\_id FROM pets WHERE pet\_name = p\_pet\_name LIMIT 1;

IF v\_pet\_id IS NULL THEN RAISE EXCEPTION 'Питомец с кличкой % не найден.', p\_pet\_name; END IF;

SELECT first\_name, last\_name INTO v\_client\_first\_name, v\_client\_last\_name FROM clients WHERE client\_id = v\_client\_id;

SELECT service\_id INTO v\_service\_id FROM services WHERE service\_name = p\_service\_name LIMIT 1;

IF v\_service\_id IS NULL THEN RAISE EXCEPTION 'Услуга с названием % не найдена.', p\_service\_name; END IF;

SELECT employee\_id, position INTO v\_vet\_id, v\_vet\_position FROM employees WHERE last\_name = p\_veterinarian\_last\_name LIMIT 1;

IF v\_vet\_id IS NULL THEN RAISE EXCEPTION 'Ветеринар с фамилией % не найден.', p\_veterinarian\_last\_name; END IF;

IF NOT EXISTS (SELECT 1 FROM vet\_service\_mapping WHERE position = v\_vet\_position AND service\_name = p\_service\_name) THEN

RAISE EXCEPTION 'Услуга % не может быть предоставлена данным ветеринаром.', p\_service\_name;

END IF;

IF NOT EXISTS (SELECT 1 FROM services WHERE service\_id = v\_service\_id AND service\_time = p\_appointment\_date) THEN

RAISE EXCEPTION 'Услуга % недоступна на указанное время.', p\_service\_name;

END IF;

IF EXISTS (SELECT 1 FROM appointments WHERE appointment\_date = p\_appointment\_date AND veterinarian\_id = v\_vet\_id) THEN

RAISE EXCEPTION 'На указанное время уже есть запись к ветеринару %.', p\_veterinarian\_last\_name;

END IF;

INSERT INTO appointments (pet\_id, service\_id, appointment\_date, complaint, veterinarian\_id, notes)

VALUES (v\_pet\_id, v\_service\_id, p\_appointment\_date, p\_complaint, v\_vet\_id, p\_notes);

RAISE NOTICE 'Запись на прием успешно добавлена для питомца % клиента % %', p\_pet\_name, v\_client\_first\_name, v\_client\_last\_name;

END;

$$;

CREATE OR REPLACE PROCEDURE update\_appointment(

p\_appointment\_id INT, p\_pet\_name VARCHAR, p\_service\_name VARCHAR,

p\_appointment\_date TIMESTAMP, p\_veterinarian\_last\_name VARCHAR,

p\_complaint TEXT, p\_notes TEXT DEFAULT NULL

)

LANGUAGE plpgsql AS $$

DECLARE

v\_pet\_id INT; v\_service\_id INT; v\_vet\_id INT; v\_existing\_appointment INT;

BEGIN

IF NOT EXISTS (SELECT 1 FROM appointments WHERE appointment\_id = p\_appointment\_id) THEN

RAISE EXCEPTION 'Запись с ID % не найдена.', p\_appointment\_id;

END IF;

SELECT pet\_id INTO v\_pet\_id FROM pets WHERE pet\_name = p\_pet\_name LIMIT 1;

IF v\_pet\_id IS NULL THEN RAISE EXCEPTION 'Питомец с кличкой % не найден.', p\_pet\_name; END IF;

SELECT service\_id INTO v\_service\_id FROM services WHERE service\_name = p\_service\_name LIMIT 1;

IF v\_service\_id IS NULL THEN RAISE EXCEPTION 'Услуга с названием % не найдена.', p\_service\_name; END IF;

SELECT employee\_id INTO v\_vet\_id FROM employees WHERE last\_name = p\_veterinarian\_last\_name LIMIT 1;

IF v\_vet\_id IS NULL THEN RAISE EXCEPTION 'Ветеринар с фамилией % не найден.', p\_veterinarian\_last\_name; END IF;

SELECT appointment\_id INTO v\_existing\_appointment FROM appointments WHERE service\_id = v\_service\_id AND veterinarian\_id = v\_vet\_id AND appointment\_date = p\_appointment\_date AND appointment\_id <> p\_appointment\_id;

IF v\_existing\_appointment IS NOT NULL THEN

RAISE EXCEPTION 'На выбранное время % уже существует запись на услугу % с данным ветеринаром.', p\_appointment\_date, p\_service\_name;

END IF;

IF NOT EXISTS (SELECT 1 FROM services WHERE service\_name = p\_service\_name AND service\_time = p\_appointment\_date AND veterinarian\_id = v\_vet\_id) THEN

RAISE EXCEPTION 'Услуга % не доступна на выбранное время для ветеринара %.', p\_service\_name, p\_veterinarian\_last\_name;

END IF;

UPDATE appointments SET pet\_id = v\_pet\_id, service\_id = v\_service\_id, appointment\_date = p\_appointment\_date,

complaint = p\_complaint, veterinarian\_id = v\_vet\_id, notes = p\_notes WHERE appointment\_id = p\_appointment\_id;

RAISE NOTICE 'Запись с ID % успешно обновлена.', p\_appointment\_id;

END;

$$;create table subject\_taught(

teacher\_id number,

subject\_id NVARCHAR2(100),

CONSTRAINT pk\_subject\_taught PRIMARY KEY (teacher\_id, subject\_id),

CONSTRAINT fk\_subject\_taught\_teacher\_id FOREIGN KEY (teacher\_id) REFERENCES teacher(id),

CONSTRAINT fk\_subject\_taught\_subject\_id FOREIGN KEY (subject\_id) REFERENCES subject(long\_name)

);

CREATE OR REPLACE PROCEDURE delete\_appointment(p\_appointment\_id INT)

LANGUAGE plpgsql AS $$

BEGIN

IF NOT EXISTS (SELECT 1 FROM appointments WHERE appointment\_id = p\_appointment\_id) THEN

RAISE EXCEPTION 'Запись на прием с ID % не найдена.', p\_appointment\_id;

END IF;

DELETE FROM appointments WHERE appointment\_id = p\_appointment\_id;

RAISE NOTICE 'Запись на прием с ID % успешно удалена.', p\_appointment\_id;

END;

$$;

Процедуры для добавления, изменения, удаления услуг для админа

CREATE OR REPLACE PROCEDURE add\_service\_for\_admin(

p\_service\_name VARCHAR, -- Имя услуги

p\_price NUMERIC, -- Цена услуги

p\_veterinarian\_last\_name VARCHAR, -- Фамилия ветеринара

p\_service\_time TIMESTAMP, -- Время услуги

p\_description TEXT DEFAULT NULL, -- Описание услуги

p\_available BOOLEAN DEFAULT TRUE -- Доступность услуги

)

LANGUAGE plpgsql AS $$

DECLARE

vet\_id INT;

vet\_position VARCHAR;

BEGIN

-- Получаем ID ветеринара

SELECT employee\_id, position INTO vet\_id, vet\_position

FROM employees

WHERE last\_name = p\_veterinarian\_last\_name

LIMIT 1;

IF vet\_id IS NULL THEN

RAISE EXCEPTION 'Ветеринар с фамилией % не найден.', p\_veterinarian\_last\_name;

END IF;

-- Проверка соответствия должности ветеринара услуге

IF NOT EXISTS (

SELECT 1

FROM vet\_service\_mapping

WHERE position = vet\_position AND service\_name = p\_service\_name

) THEN

RAISE EXCEPTION 'Услуга % не может быть предоставлена данным ветеринаром, так как его должность (%), не соответствует данной услуге.', p\_service\_name, vet\_position;

END IF;

-- Проверка времени услуги на уникальность для того же ветеринара

IF EXISTS (

SELECT 1

FROM services

WHERE veterinarian\_id = vet\_id

AND service\_time = p\_service\_time

) THEN

RAISE EXCEPTION 'Услуга с таким временем (%), уже существует для данного ветеринара.', p\_service\_time;

END IF;

-- Проверка времени услуги

IF p\_service\_time < CURRENT\_DATE THEN

RAISE EXCEPTION 'Дата услуги должна быть не раньше сегодняшнего дня.';

END IF;

-- Проверка на корректность описания (только буквы и пробелы)

IF p\_description IS NOT NULL AND p\_description !~ '^[А-Яа-яA-Za-z ]+$' THEN

RAISE EXCEPTION 'Описание услуги должно содержать только буквы и пробелы.';

END IF;

-- Вставка услуги

INSERT INTO services (service\_name, price, veterinarian\_id, veterinarian\_first\_name, veterinarian\_last\_name, service\_time, available, description)

VALUES (p\_service\_name, p\_price, vet\_id, (SELECT first\_name FROM employees WHERE employee\_id = vet\_id), p\_veterinarian\_last\_name, p\_service\_time, p\_available, p\_description);

RAISE NOTICE 'Услуга % успешно добавлена для ветеринара %', p\_service\_name, p\_veterinarian\_last\_name;

END;

$$;

CREATE OR REPLACE PROCEDURE update\_service\_for\_admin(

p\_service\_id INT, -- ID услуги

p\_service\_name VARCHAR, -- Имя услуги

p\_price NUMERIC, -- Цена услуги

p\_veterinarian\_last\_name VARCHAR, -- Фамилия ветеринара

p\_service\_time TIMESTAMP, -- Время услуги

p\_description TEXT DEFAULT NULL, -- Описание услуги

p\_available BOOLEAN DEFAULT TRUE -- Доступность услуги

)

LANGUAGE plpgsql AS $$

DECLARE

vet\_id INT;

BEGIN

-- Получаем ID ветеринара

SELECT employee\_id INTO vet\_id

FROM employees

WHERE last\_name = p\_veterinarian\_last\_name

LIMIT 1;

IF vet\_id IS NULL THEN

RAISE EXCEPTION 'Ветеринар с фамилией % не найден.', p\_veterinarian\_last\_name;

END IF;

-- Проверка времени услуги на уникальность для того же ветеринара (для обновлений)

IF EXISTS (

SELECT 1

FROM services

WHERE veterinarian\_id = vet\_id

AND service\_time = p\_service\_time

AND service\_id != p\_service\_id

) THEN

RAISE EXCEPTION 'Услуга с таким временем (%), уже существует для данного ветеринара.', p\_service\_time;

END IF;

-- Проверка на корректность описания (только буквы и пробелы)

IF p\_description IS NOT NULL AND p\_description !~ '^[А-Яа-яA-Za-z ]+$' THEN

RAISE EXCEPTION 'Описание услуги должно содержать только буквы и пробелы.';

END IF;

-- Обновление услуги

UPDATE services

SET service\_name = p\_service\_name,

price = p\_price,

veterinarian\_id = vet\_id,

veterinarian\_first\_name = (SELECT first\_name FROM employees WHERE employee\_id = vet\_id),

veterinarian\_last\_name = p\_veterinarian\_last\_name,

service\_time = p\_service\_time,

available = p\_available,

description = p\_description

WHERE service\_id = p\_service\_id;

RAISE NOTICE 'Услуга с ID % успешно обновлена.', p\_service\_id;

END;

$$;

CREATE OR REPLACE PROCEDURE delete\_service\_for\_admin(

p\_service\_id INT -- ID услуги для удаления

)

LANGUAGE plpgsql AS $$

BEGIN

-- Удаление услуги

DELETE FROM services WHERE service\_id = p\_service\_id;

RAISE NOTICE 'Услуга с ID % успешно удалена.', p\_service\_id;

END;

$$;

Процедуры для добавления, изменения, удаления животных для админа

CREATE OR REPLACE PROCEDURE add\_pet\_for\_admin(

p\_pet\_name VARCHAR, -- Имя питомца

p\_species VARCHAR, -- Вид питомца

p\_breed VARCHAR, -- Порода питомца

p\_birth\_date DATE, -- Дата рождения питомца

p\_username VARCHAR -- Логин пользователя, для которого добавляется питомец

)

LANGUAGE plpgsql AS $$

DECLARE

p\_owner\_id INT; -- Переменная для хранения ID владельца

valid\_breeds TEXT[]; -- Массив допустимых пород для указанного вида питомца

current\_date DATE := CURRENT\_DATE; -- Текущая дата

BEGIN

-- Проверка на формат даты рождения

IF p\_birth\_date::TEXT !~ '^\d{4}-\d{2}-\d{2}$' THEN

RAISE EXCEPTION 'Неверный формат даты рождения. Ожидается формат yyyy-mm-dd.';

END IF;

-- Получаем ID владельца по логину пользователя

SELECT c.client\_id INTO p\_owner\_id

FROM clients c

JOIN users u ON c.user\_id = u.user\_id

WHERE u.username = p\_username

LIMIT 1;

-- Проверка на существование владельца

IF p\_owner\_id IS NULL THEN

RAISE EXCEPTION 'Владелец с логином % не найден.', p\_username;

END IF;

-- Валидация имени питомца

IF p\_pet\_name IS NULL OR LENGTH(TRIM(p\_pet\_name)) < 3 THEN

RAISE EXCEPTION 'Имя питомца должно содержать минимум 3 символа.';

END IF;

IF NOT p\_pet\_name ~\* '^[А-Яа-яA-Za-z ]+$' THEN

RAISE EXCEPTION 'Имя питомца должно содержать только буквы.';

END IF;

-- Валидация даты рождения

IF p\_birth\_date > current\_date THEN

RAISE EXCEPTION 'Дата рождения не может быть в будущем.';

ELSIF EXTRACT(YEAR FROM p\_birth\_date) < 1000 THEN

RAISE EXCEPTION 'Дата рождения не может быть раньше 1000 года.';

END IF;

-- Валидация вида питомца и пород

CASE p\_species

WHEN 'Собака' THEN valid\_breeds := ARRAY['Лабрадор', 'Немецкая овчарка'];

WHEN 'Кошка' THEN valid\_breeds := ARRAY['Сиамская', 'Персидская', 'Британец'];

WHEN 'Попугай' THEN valid\_breeds := ARRAY['Ара', 'Жако'];

WHEN 'Хомяк' THEN valid\_breeds := ARRAY['Сирийский', 'Джунгарский'];

WHEN 'Кролик' THEN valid\_breeds := ARRAY['Мини-ломброза', 'Ангорский'];

ELSE

RAISE EXCEPTION 'Вид питомца должен быть одним из следующих: Собака, Кошка, Попугай, Хомяк, Кролик.';

END CASE;

-- Проверка на допустимость породы для выбранного вида

IF p\_breed IS NOT NULL AND array\_position(valid\_breeds, p\_breed::text) IS NULL THEN

RAISE EXCEPTION 'Порода % не подходит для вида питомца %.', p\_breed, p\_species;

END IF;

-- Добавление питомца

INSERT INTO pets (pet\_name, species, breed, birth\_date, owner\_id)

VALUES (p\_pet\_name, p\_species, p\_breed, p\_birth\_date, p\_owner\_id);

RAISE NOTICE 'Питомец % успешно добавлен владельцу с логином %.', p\_pet\_name, p\_username;

END;

$$;

CREATE OR REPLACE PROCEDURE update\_pet\_for\_admin(

p\_pet\_name VARCHAR, -- Имя питомца, которое нужно обновить

p\_username VARCHAR, -- Логин владельца питомца

p\_new\_pet\_name VARCHAR DEFAULT NULL, -- Новое имя питомца (если нужно)

p\_new\_species VARCHAR DEFAULT NULL, -- Новый вид питомца (если нужно)

p\_new\_breed VARCHAR DEFAULT NULL, -- Новая порода питомца (если нужно)

p\_new\_birth\_date DATE DEFAULT NULL -- Новая дата рождения питомца

)

LANGUAGE plpgsql AS $$

DECLARE

p\_owner\_id INT; -- Переменная для хранения ID владельца

current\_date DATE := CURRENT\_DATE; -- Текущая дата

valid\_breeds TEXT[]; -- Массив допустимых пород для указанного вида питомца

current\_species VARCHAR; -- Текущий вид питомца

BEGIN

-- Проверка на формат новой даты рождения

IF p\_new\_birth\_date IS NOT NULL AND p\_new\_birth\_date::TEXT !~ '^\d{4}-\d{2}-\d{2}$' THEN

RAISE EXCEPTION 'Неверный формат новой даты рождения. Ожидается формат yyyy-mm-dd.';

END IF;

-- Получаем ID владельца питомца по логину

SELECT c.client\_id, p.species INTO p\_owner\_id, current\_species

FROM clients c

JOIN users u ON c.user\_id = u.user\_id

JOIN pets p ON p.owner\_id = c.client\_id

WHERE u.username = p\_username AND p.pet\_name = p\_pet\_name

LIMIT 1;

-- Проверка на существование владельца и питомца

IF p\_owner\_id IS NULL THEN

RAISE EXCEPTION 'Питомец с именем % не найден для владельца с логином %.', p\_pet\_name, p\_username;

END IF;

-- Валидация новых значений (если они не NULL)

IF p\_new\_pet\_name IS NOT NULL THEN

IF LENGTH(TRIM(p\_new\_pet\_name)) < 3 THEN

RAISE EXCEPTION 'Новое имя питомца должно содержать минимум 3 символа.';

END IF;

IF NOT p\_new\_pet\_name ~\* '^[А-Яа-яA-Za-z ]+$' THEN

RAISE EXCEPTION 'Новое имя питомца должно содержать только буквы.';

END IF;

END IF;

IF p\_new\_birth\_date IS NOT NULL THEN

IF p\_new\_birth\_date > current\_date THEN

RAISE EXCEPTION 'Дата рождения не может быть в будущем.';

END IF;

IF EXTRACT(YEAR FROM p\_new\_birth\_date) < 1000 THEN

RAISE EXCEPTION 'Дата рождения не может быть раньше 1000 года.';

END IF;

END IF;

-- Обновление питомца

UPDATE pets

SET

pet\_name = COALESCE(p\_new\_pet\_name, pet\_name),

species = COALESCE(p\_new\_species, species),

breed = COALESCE(p\_new\_breed, breed),

birth\_date = COALESCE(p\_new\_birth\_date, birth\_date)

WHERE pet\_name = p\_pet\_name AND owner\_id = p\_owner\_id;

RAISE NOTICE 'Питомец % успешно обновлен владельцу с логином %.', p\_pet\_name, p\_username;

END;

$$;

CREATE OR REPLACE PROCEDURE delete\_pet\_for\_admin(p\_pet\_name VARCHAR, p\_username VARCHAR)

LANGUAGE plpgsql AS $$

DECLARE

p\_owner\_id INT;

BEGIN

-- Получаем ID владельца по логину пользователя

SELECT c.client\_id INTO p\_owner\_id

FROM clients c

JOIN users u ON c.user\_id = u.user\_id

WHERE u.username = p\_username

LIMIT 1;

-- Проверка на существование владельца

IF p\_owner\_id IS NULL THEN

RAISE EXCEPTION 'Владелец с логином % не найден.', p\_username;

END IF;

-- Проверка на существование питомца у владельца

IF NOT EXISTS (SELECT 1 FROM pets WHERE pet\_name = p\_pet\_name AND owner\_id = p\_owner\_id) THEN

RAISE EXCEPTION 'Питомец с именем % не найден для владельца %.', p\_pet\_name, p\_username;

END IF;

-- Удаление питомца

DELETE FROM pets WHERE pet\_name = p\_pet\_name AND owner\_id = p\_owner\_id;

RAISE NOTICE 'Питомец с именем % успешно удален владельцу с логином %.', p\_pet\_name, p\_username;

END;

$$;

Процедуры добавления, изменения, удаления рецептов для админа

CREATE OR REPLACE PROCEDURE add\_prescription\_by\_pet\_name(

p\_pet\_name VARCHAR, -- Имя питомца

p\_veterinarian\_name VARCHAR, -- Имя и фамилия ветеринара

p\_medication\_name VARCHAR, -- Название лекарства

p\_dosage VARCHAR, -- Дозировка

p\_administration\_method VARCHAR, -- Способ применения

p\_treatment\_duration VARCHAR, -- Продолжительность лечения

p\_notes TEXT DEFAULT NULL -- Примечания

)

LANGUAGE plpgsql AS $$

DECLARE

v\_pet\_id INT;

v\_veterinarian\_id INT;

v\_diagnosis\_id INT;

BEGIN

-- Получение ID питомца

SELECT pet\_id INTO v\_pet\_id

FROM pets

WHERE pet\_name = p\_pet\_name

LIMIT 1;

IF v\_pet\_id IS NULL THEN

RAISE EXCEPTION 'Питомец с именем % не найден.', p\_pet\_name;

END IF;

-- Получение ID ветеринара

SELECT employee\_id INTO v\_veterinarian\_id

FROM employees

WHERE CONCAT(first\_name, ' ', last\_name) = p\_veterinarian\_name

LIMIT 1;

IF v\_veterinarian\_id IS NULL THEN

RAISE EXCEPTION 'Ветеринар с именем % не найден.', p\_veterinarian\_name;

END IF;

-- Проверка наличия диагноза

SELECT diagnosis\_id INTO v\_diagnosis\_id

FROM diagnoses

WHERE pet\_id = v\_pet\_id AND veterinarian\_id = v\_veterinarian\_id

ORDER BY diagnosis\_date DESC

LIMIT 1;

IF v\_diagnosis\_id IS NULL THEN

RAISE EXCEPTION 'Диагноз для питомца % не найден.', p\_pet\_name;

END IF;

-- Вставка рецепта

INSERT INTO prescriptions (

pet\_id, veterinarian\_id, diagnosis\_id, prescription\_date,

medication\_name, dosage, administration\_method,

treatment\_duration, notes

) VALUES (

v\_pet\_id, v\_veterinarian\_id, v\_diagnosis\_id, NOW(),

p\_medication\_name, p\_dosage, p\_administration\_method,

p\_treatment\_duration, p\_notes

);

RAISE NOTICE 'Рецепт для питомца % успешно добавлен.', p\_pet\_name;

END;

$$;

CREATE OR REPLACE PROCEDURE update\_prescription\_by\_pet\_name(

p\_pet\_name VARCHAR,

p\_veterinarian\_name VARCHAR,

p\_medication\_name VARCHAR,

p\_new\_dosage VARCHAR,

p\_new\_administration\_method VARCHAR,

p\_new\_treatment\_duration VARCHAR,

p\_new\_notes TEXT DEFAULT NULL

)

LANGUAGE plpgsql AS $$

DECLARE

v\_pet\_id INT;

v\_veterinarian\_id INT;

v\_prescription\_id INT;

BEGIN

-- Получение ID питомца

SELECT pet\_id INTO v\_pet\_id

FROM pets

WHERE pet\_name = p\_pet\_name

LIMIT 1;

IF v\_pet\_id IS NULL THEN

RAISE EXCEPTION 'Питомец с именем % не найден.', p\_pet\_name;

END IF;

-- Получение ID ветеринара

SELECT employee\_id INTO v\_veterinarian\_id

FROM employees

WHERE CONCAT(first\_name, ' ', last\_name) = p\_veterinarian\_name

LIMIT 1;

IF v\_veterinarian\_id IS NULL THEN

RAISE EXCEPTION 'Ветеринар с именем % не найден.', p\_veterinarian\_name;

END IF;

-- Получение ID рецепта

SELECT prescription\_id INTO v\_prescription\_id

FROM prescriptions

WHERE pet\_id = v\_pet\_id AND veterinarian\_id = v\_veterinarian\_id

AND medication\_name = p\_medication\_name

ORDER BY prescription\_date DESC

LIMIT 1;

IF v\_prescription\_id IS NULL THEN

RAISE EXCEPTION 'Рецепт для питомца % не найден.', p\_pet\_name;

END IF;

-- Обновление рецепта

UPDATE prescriptions

SET

dosage = p\_new\_dosage,

administration\_method = p\_new\_administration\_method,

treatment\_duration = p\_new\_treatment\_duration,

notes = p\_new\_notes

WHERE prescription\_id = v\_prescription\_id;

RAISE NOTICE 'Рецепт для питомца % успешно обновлен.', p\_pet\_name;

END;

$$;

CREATE OR REPLACE PROCEDURE delete\_prescription\_by\_pet\_name(

p\_pet\_name VARCHAR,

p\_veterinarian\_name VARCHAR,

p\_medication\_name VARCHAR

)

LANGUAGE plpgsql AS $$

DECLARE

v\_pet\_id INT;

v\_veterinarian\_id INT;

v\_prescription\_id INT;

BEGIN

-- Получение ID питомца

SELECT pet\_id INTO v\_pet\_id

FROM pets

WHERE pet\_name = p\_pet\_name

LIMIT 1;

IF v\_pet\_id IS NULL THEN

RAISE EXCEPTION 'Питомец с именем % не найден.', p\_pet\_name;

END IF;

-- Получение ID ветеринара

SELECT employee\_id INTO v\_veterinarian\_id

FROM employees

WHERE CONCAT(first\_name, ' ', last\_name) = p\_veterinarian\_name

LIMIT 1;

IF v\_veterinarian\_id IS NULL THEN

RAISE EXCEPTION 'Ветеринар с именем % не найден.', p\_veterinarian\_name;

END IF;

-- Получение ID рецепта

SELECT prescription\_id INTO v\_prescription\_id

FROM prescriptions

WHERE pet\_id = v\_pet\_id AND veterinarian\_id = v\_veterinarian\_id

AND medication\_name = p\_medication\_name

ORDER BY prescription\_date DESC

LIMIT 1;

IF v\_prescription\_id IS NULL THEN

RAISE EXCEPTION 'Рецепт для питомца % не найден.', p\_pet\_name;

END IF;

-- Удаление рецепта

DELETE FROM prescriptions

WHERE prescription\_id = v\_prescription\_id;

RAISE NOTICE 'Рецепт для питомца % успешно удален.', p\_pet\_name;

END;

$$;

Приложение Б

Процедура импорта

|  |
| --- |
| CREATE OR REPLACE PROCEDURE import\_reviews\_from\_json(p\_file\_path text)  LANGUAGE plpgsql  AS $$  BEGIN  -- Создание временной таблицы для хранения данных  CREATE TEMP TABLE temp\_reviews (data json);    -- Импорт данных из файла  EXECUTE format('COPY temp\_reviews FROM %L', p\_file\_path);    -- Вставка данных из временной таблицы в таблицу reviews  INSERT INTO reviews (client\_id, service\_id, review\_date, rating, review\_text, response\_text)  SELECT  (value->>'client\_id')::int AS client\_id,  (value->>'service\_id')::int AS service\_id,  (value->>'review\_date')::timestamp AS review\_date,  (value->>'rating')::int AS rating,  value->>'review\_text' AS review\_text,  value->>'response\_text' AS response\_text  FROM temp\_reviews, json\_array\_elements(data) AS value;  -- Удаление временной таблицы  DROP TABLE IF EXISTS temp\_reviews;  END;  $$; |