



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт (Филиал) № 8 «Компьютерные науки и прикладная математика» Кафедра 806
Группа М8О-406Б-19 Направление подготовки 01.03.02 «Прикладная математика и
информатика»

Профиль Информатика

Квалификация: бакалавр

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

На тему: Разработка базы данных отделения неотложной медицинской помощи
с использованием PostgreSQL

Автор ВКРБ:	Алексеев Владислав Евгеньевич	(_____)
Руководитель:	Пивоваров Дмитрий Евгеньевич	(_____)
Консультант:	—	(_____)
Консультант:	—	(_____)
Рецензент:	—	(_____)

К защите допустить

Заведующий кафедрой № 806	Крылов Сергей Сергеевич	(_____)
_____ мая 2023 года		

РЕФЕРАТ

Выпускная квалификационная работа бакалавра состоит из 76 страниц, 47 рисунков, 20 использованных источников, 2 приложений.

БАЗА ДАННЫХ, АВТОМАТИЗАЦИЯ, БЕЗОПАСНОСТЬ, ШИФРОВАНИЕ, POSTGRESQL

Объектом исследования в данной выпускной квалификационной работе является деятельность отделения неотложной медицинской помощи.

Предметом исследования является процесс разработки и внедрения базы данных медицинских диагностических таблиц первичного осмотра пациента, таблиц дифференциальной диагностики и вспомогательных информационных таблиц.

Целью работы является автоматизация, ускорение и улучшение условий обработки данных для сотрудников отделения неотложной медицинской помощи.

Основное содержание работы состояло в разработке базы данных медицинских диагностических таблиц первичного осмотра пациента для сотрудников отделения неотложной медицинской помощи.

Для достижения поставленной цели был проведен анализ функциональных требований для разработки необходимой базы данных, а также анализ наиболее эффективных способов хранения взаимосвязанных атрибутов каждой сущности. Кроме того, была разработана и протестирована довольно обширная структура базы данных, объединяющая сильные стороны проанализированных методов хранения и обработки информации.

Основными результатами работы, полученными в процессе анализа и разработки, являются: определение наиболее эффективных способов структурирования и хранения всей необходимой информации для работников скорой медицинской помощи, разработка и внедрение базы данных, включая вспомогательные информационные таблицы, оценка эффективной работы полученной базы данных и выявление сценариев ее оптимального использования.

Полученные результаты разработки представляют немаловажное значение для всех организаций здравоохранения, стремящихся оптимизировать работу своего персонала и минимизировать время и трудозатраты на рутинное заполнение бумажных документов.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	4
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ БАЗЫ ДАННЫХ	9
1.1 Анализ предметной области	9
1.2 Анализ существующих подходов	12
1.3 Назначение и возможности базы данных	15
2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА	19
2.1 Выбор программных средств для реализации базы данных	19
2.2 Проектирование базы данных	20
2.3 Проектирование логических моделей данных	23
2.4 Нормализация базы данных	24
2.5 Проектирование физических моделей данных	25
2.6 Готовые решения	26
3 ЗАЩИТА БАЗЫ ДАННЫХ ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА	34
3.1 Основные угрозы и уязвимости	34
3.2 Конфиденциальность персональных данных	35
3.3 Использование имен пользователей, ролей и разрешений	37
3.4 Шифрование базы данных	41
3.5 Порты	47
3.6 Брандмауэр	48
4 ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ	50
4.1 Методы и инструменты тестирования	50
4.2 Нагрузочное тестирование и оптимизация производительности	50
4.3 Выполнение запросов	56
ЗАКЛЮЧЕНИЕ	65
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	66
ПРИЛОЖЕНИЕ А Исходный код	68
ПРИЛОЖЕНИЕ Б Нагрузочное тестирование	69

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе бакалавра применяют следующие термины с соответствующими определениями:

Модель OSI — это сетевая модель стека сетевых протоколов, посредством которой различные сетевые устройства могут взаимодействовать друг с другом

Наряд — это группа медицинских работников и транспортных средств, назначенных для выполнения определенной миссии или задачи, а также медицинское оборудование и инструменты

ACID — это набор свойств транзакций базы данных, предназначенных для гарантии достоверности данных, несмотря на ошибки, сбои питания и другие неудачи

BSON — это формат электронного обмена цифровыми данными, бинарная форма представления простых структур данных и ассоциативных массивов. Является надмножеством JSON, включая дополнительно регулярные выражения, двоичные данные и даты

JSON — это популярный формат текстовых данных, который используется для обмена данными в современных веб - и мобильных приложениях

SQL-инъекция — это уязвимость, которая позволяет атакующему использовать фрагмент вредоносного кода на языке структурированных запросов (SQL) для манипулирования базой данных и получения доступа к потенциально ценной информации

SSL — это криптографический протокол, который подразумевает более безопасную связь и использует асимметричную криптографию для аутентификации ключей обмена, симметричное шифрование для сохранения конфиденциальности, коды аутентификации сообщений для целостности сообщений

TCP — это важный протокол сети интернет, который позволяет двум хостам создать соединение и обмениваться потоками данных

UDP — это один из ключевых элементов набора сетевых протоколов для Интернета, с помощью которого компьютерные приложения могут посыпать сообщения другим хостам по IP-сети без необходимости предварительного сообщения для установки специальных каналов передачи или путей данных

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе бакалавра применяют следующие сокращения и обозначения:

БД — база данных

ИС — информационная система

МКБ — международная классификация болезней

НСД — несанкционированный доступ

ОНМП — отделение неотложной медицинской помощи

ОС — операционная система

СМП — скорая медицинская помощь

СУБД — система управления базами данных

ACID — атомарность, согласованность, изоляция и долговечность

ANSI — американский национальный институт стандартов

OSI — открытые системы взаимодействия

TCP — протокол управления передачей

UDP — протокол пользовательских датаграмм

XML — расширяемый язык разметки

ВВЕДЕНИЕ

Актуальность темы данной выпускной квалификационной работы бакалавра связана с большим значением проведения эффективных мероприятия по оказанию медицинской помощи пациентам со стороны медицинских работников, максимальное упрощение заполнения документов о выездах и действиях по ее оказанию. Кроме того, врачам иногда требуется некоторый вспомогательный материал в виде информационных данных в таблицах с симптомами диагнозов или дозировками того или иного препарата.

В наше время использование баз данных является одной из неотъемлемых частей работы многих компаний и организаций, так как это позволяет хранить, организовывать и управлять большими объемами данных. Базы данных позволяют быстро и эффективно хранить и обрабатывать данные, что является необходимым условием для принятия важных бизнес-решений.

Одним из наиболее популярных языков для работы с базами данных является PostgreSQL. Он обладает высокой степенью надежности, масштабируемости и производительности. PostgreSQL также поддерживает многие функции, которые делают его удобным для работы с различными типами данных и приложений [1].

Использование баз данных также позволяет легко и быстро находить необходимую информацию и управлять ею, а также обеспечивает сохранность и целостность данных, что немаловажно в наших современных реалиях, ведь большинство информации просто напросто может утечь в сеть и оказаться в открытом доступе, желаем мы того или нет. Базы данных позволяют использовать различные методы анализа и обработки данных, что является важным инструментом для повышения эффективности работы компаний и организаций.

Кроме того, использование баз данных является современным и необходимым подходом к работе с данными, который позволяет не только сохранять и управлять ими, но и анализировать и использовать в дальнейшем. Базы данных PostgreSQL являются открытым и свободно распространяемым решением, что делает их доступными и удобными для использования в различных проектах и приложениях.

В современном мире качество медицинской помощи и скорость ее

оказания имеют большое значение для общества. Скорая медицинская помощь играет важную роль в сохранении здоровья и жизни людей в случае непредвиденных ситуаций и аварий. Однако, существующая система скорой помощи имеет свои проблемы, связанные с низкой эффективностью и долгим временем ожидания при оказании медицинской помощи.

Внедрение цифровых технологий в работу скорой медицинской помощи может значительно улучшить качество и эффективность ее работы. В частности, организация электронных медицинских карт и использование баз данных на языке PostgreSQL позволят ускорить процесс диагностики и лечения пациентов, а также сократить время ожидания скорой медицинской помощи.

Данная выпускная квалификационная работа имеет практическое значение, так как внедрение цифровых технологий в работу скорой медицинской помощи может ускорить процесс оказания медицинских услуг и повысить их качество, что в свою очередь приведет к улучшению уровня здоровья и жизни населения в целом.

Сейчас врачам важно и нужно переходить на цифровое использование и заполнение медицинских карт при осмотре пациентов по нескольким причинам:

- цифровая медицинская карта позволяет быстро и удобно получить доступ к информации о пациенте, что повышает эффективность и точность диагностики и лечения;
- цифровая медицинская карта позволяет сохранять и обрабатывать большие объемы данных о пациентах, что является важным инструментом для проведения исследований и разработки новых методов лечения. Также это позволяет врачам получать доступ к общей истории лечения пациента и учитывать его предыдущие заболевания, что повышает качество и эффективность лечения;
- использование цифровых медицинских карт позволяет уменьшить вероятность ошибок и искажений при заполнении и хранении информации о пациентах, что обеспечивает сохранность и конфиденциальность медицинских данных.

Наконец, использование цифровых медицинских карт является современным и удобным подходом к организации работы медицинских учреждений и обслуживанию пациентов, что способствует повышению

уровня медицинской помощи и общей качества жизни населения.

В данной выпускной квалификационной работе бакалавра были использованы различные технологии и инструменты для облегчения исследования, реализации и анализа работы всей системы:

- PostgreSQL – мощная реляционная система управления базами данных (СУБД), которая обладает высокой производительностью, масштабируемостью, надежностью и расширяемостью. Она поддерживает широкий спектр функций, позволяет работать с различными типами данных и обеспечивает сохранность и целостность данных;
- pgAdmin – это графический инструмент для администрирования баз данных PostgreSQL. Он предоставляет удобный интерфейс для управления базами данных и объектами в них, такими как таблицы, индексы, пользователи и многое другое. pgAdmin поддерживает широкий диапазон функций, включая создание и редактирование объектов базы данных, выполнение SQL-запросов, экспорт и импорт данных, а также управление безопасностью и настройками базы данных;
- Erwin – это программное обеспечение для моделирования баз данных. Оно позволяет проектировать их визуально в удобном графическом интерфейсе. Erwin поддерживает множество популярных СУБД, включая PostgreSQL, и предоставляет широкий набор инструментов для работы с базами данных, таких как автоматическое генерирование SQL-кода, проверка целостности данных, анализ производительности и многое другое.

С научной точки зрения, данная работа вносит вклад в существующий массив знаний о разработке и проектировании баз данных и их эффективности в различных сценариях нагрузки и использования. С практической точки зрения, данное исследование предоставляет ценные идеи для медицинских работников, для которых так важно оптимизировать свои действия по заполнению медицинских карт, карт осмотра пациентов и многие другие рутинные задачи.

Таким образом, данная выпускная квалификационная работа бакалавра является актуальной и с научно-методической/теоретической, и с практической точек зрения.

1 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ БАЗЫ ДАННЫХ

1.1 Анализ предметной области

Анализ предметной области является первым этапом в проектировании базы данных, в ходе которого выделяются основные объекты и их свойства, определяются первоначальные требования и границы проекта, чтобы разработать эффективную и безопасную базу данных.

Предметной областью разрабатываемой модели данных является система заполнения электронных медицинских карт для отделения неотложной медицинской помощи.

Скорая медицинская помощь (СМП) – вид медицинской помощи, оказываемой гражданам при заболеваниях, несчастных случаях, травмах, отравлениях и других состояниях, требующих срочного медицинского вмешательства.

Отделение неотложной (экстренной) медицинской помощи больницы является структурным подразделением многопрофильной больницы, которое в круглосуточном режиме оказывает экстренную (неотложную) медицинскую помощь.

Медицинская помощь включает в себя процедуру проведения осмотра, непосредственно манипуляции по оказанию медицинской помощи, а также консультирование пациента по с целью определения наиболее эффективного, безопасного и экономически оправданного курса лечения.

ОНМП осуществляет следующие функции:

- прием пациентов с острыми заболеваниями, несчастными случаями, травмами, отравлениями и другими состояниями, требующими немедленной медицинской помощи;
- проведение первичной медицинской диагностики и оценки состояния пациента, осуществление мер, направленных на стабилизацию состояния пациента;
- предоставление неотложной медицинской помощи, включая проведение лечебных манипуляций, инъекций, переливание крови, а также оказание психологической помощи;
- организация и координация работы других специалистов и служб в медицинском учреждении, если необходимо;

- подготовка пациента к транспортировке в стационар для дальнейшего лечения;
- соблюдение всех медицинских стандартов и требований, направленных на обеспечение безопасности пациентов и медицинского персонала;
- организация транспортировки пациентов в случае необходимости;
- обеспечение работы необходимого медицинского оборудования, инструментов, материалов и медикаментов, необходимых для оказания неотложной медицинской помощи,
- проведение медицинской документации, включая учет медицинских случаев, регистрацию историй болезни и медицинских записей;
- обеспечение мониторинга и контроля за состоянием пациентов, находящихся на лечении в отделении неотложной (экстренной) медицинской помощи;
- обучение медицинского персонала и сотрудников отделения неотложной медицинской помощи новым методам лечения и диагностики;
- сотрудничество и консультации с другими медицинскими учреждениями, специалистами и службами для повышения качества оказываемой медицинской помощи.

Отделение неотложной (экстренной) медицинской помощи является ключевым звеном в системе оказания медицинской помощи населению и обеспечивает быстрое и эффективное лечение при острых заболеваниях и травмах.

Базы данных для неотложной медицинской помощи являются критически важным элементом в работе современных медицинских учреждений. Они позволяют эффективно организовывать, хранить и обрабатывать медицинскую информацию, ускоряя процессы принятия решений и повышая качество медицинской помощи.

Основным требованием к базе данных для неотложной медицинской помощи является ее способность оперативно и точно хранить медицинскую информацию о пациентах. Эта информация может включать данные о медицинской истории пациента, диагнозе, принятых мероприятиях, результатах обследований и лекарственном лечении.

Для эффективного управления медицинской информацией в базе

данных неотложной медицинской помощи необходима специализированная система управления базами данных (СУБД). Существует множество СУБД, которые могут использоваться для этой цели, включая Oracle, MySQL, Microsoft SQL Server, PostgreSQL и др.

Также необходимо учитывать специфику работы медицинских учреждений, которые могут иметь различные требования к хранению и обработке медицинской информации. Например, больницы могут иметь разные отделения, каждое из которых может иметь свои особенности в обработке информации. Кроме того, необходимо учитывать возможность интеграции с другими системами, такими как системы управления ресурсами и планирования процессов.

Важно также отметить, что база данных для неотложной медицинской помощи должна соответствовать требованиям законодательства в области защиты персональных данных, таких как Федеральный закон от 27 июля 2006 года № 152-ФЗ «О персональных данных». Регулярное обновление и адаптация базы данных к изменениям в законодательстве помогут поддерживать соответствие нормам и предотвращать правовые проблемы.

Исследования показывают, что эффективное использование баз данных в медицинских учреждениях может значительно улучшить качество медицинской помощи и снизить затраты на ее оказание. Например, использование баз данных может уменьшить количество ошибок в принятии решений, сократить время, необходимое для оказания медицинской помощи, и повысить точность диагностики.

На текущий момент существует множество различных подходов к проектированию баз данных для медицинских учреждений. Некоторые из них ориентированы на сохранение структурированной информации, другие на работу с полуструктуризованными и неструктуризованными данными.

Одним из наиболее распространенных подходов к проектированию баз данных для медицинских учреждений является использование реляционной модели данных [2]. Реляционная модель данных позволяет хранить информацию в виде таблиц, которые могут быть связаны друг с другом через ключи. Это позволяет обрабатывать большие объемы структурированных данных и осуществлять запросы на выборку данных, используя язык SQL.

Тем не менее, в последние годы набирают популярность нереляционные базы данных, такие как MongoDB, Cassandra, Couchbase и др. Они хранят

данные в более гибкой форме, позволяют более легко масштабировать базы данных и работать с полуструктурированными и неструктурированными данными, такими как тексты медицинских записей, изображения и видео.

Важным аспектом разработки баз данных для медицинских учреждений является также обеспечение защиты медицинской информации от несанкционированного доступа. Для этого применяются различные меры безопасности, такие как шифрование данных, многофакторная аутентификация и разграничение прав доступа к информации в зависимости от роли пользователя.

Наконец, важно отметить, что разработка баз данных для медицинских учреждений является сложным и многогранным процессом, требующим учета множества факторов, таких как специфика медицинских процедур, законодательство, требования к безопасности и другие. Поэтому необходимо проводить тщательный анализ предметной области и разрабатывать индивидуальные решения для каждого конкретного медицинского учреждения.

1.2 Анализ существующих подходов

Перед тем как начинать разработку базы данных, нужно выяснить, какие виды БД существуют, проанализировать их и уже на основе этого делать выводы о наиболее подходящем виде под наши конкретные требования.

Существует несколько видов баз данных, каждый из которых имеет свои особенности и применения:

- реляционные базы данных (РБД): это самый распространенный тип БД. В РБД данные организованы в виде таблиц, состоящих из строк (записей) и столбцов (полей). Реляционные БД используют структурированный язык запросов, такой как SQL, для управления данными и выполнения операций, таких как вставка, обновление, удаление и извлечение данных. Примеры реляционных БД включают MySQL, PostgreSQL и Oracle;
- иерархические базы данных: в таких БД данные организованы в виде иерархической структуры, состоящей из уровней и подуровней. Каждый уровень может иметь только одного родителя, что создает иерархию. Этот тип БД широко использовался в прошлом, но в настоящее время его применение ограничено.

Примеры иерархических БД включают IBM's Information Management System (IMS);

- сетевые базы данных: этот тип БД организован в виде сети связанных записей. Записи могут иметь несколько родителей и несколько дочерних записей, что позволяет создавать сложные связи между данными. Сетевые БД также имеют ограниченное применение в настоящее время. Примером сетевой БД является Integrated Data Store (IDS);
- объектно-ориентированные базы данных (ООБД): в ООБД данные организованы в виде объектов, которые могут содержать свойства (поля) и методы (функции). ООБД позволяют более натуральное представление сложных структур данных и поддерживают наследование и полиморфизм. Примеры ООБД включают MongoDB и Couchbase;
- ключ-значение базы данных: в таких БД данные хранятся в виде пар ключ-значение. Они просты и эффективны для хранения и извлечения данных, но ограничены по функциональности. Примеры ключ-значение БД включают Redis и Apache Cassandra;
- документоориентированные базы данных: в этом типе БД данные организованы в виде документов, обычно в формате JSON или XML. Документы могут содержать различные поля и вкладываться друг в друга, образуя более сложную структуру;
- временные базы данных: эти базы данных предназначены для хранения и управления временными данными, такими как временные ряды, события и временные маркировки. Они оптимизированы для выполнения операций, связанных с временными данными, такими как агрегация, интерполяция и анализ трендов. Примеры временных баз данных включают InfluxDB и TimescaleDB;
- массивные параллельные обработки (МРР) базы данных: эти базы данных разработаны для обработки больших объемов данных с использованием параллельных вычислений на распределенных системах. Они позволяют выполнить параллельную обработку запросов и аналитики на большом количестве узлов. Примеры МРР БД включают Amazon Redshift и Google BigQuery.

Каждый из этих видов БД предназначен для решения определенных задач и имеет свои преимущества и ограничения. Выбор типа базы данных зависит от конкретных требований проекта, характеристик данных и ожидаемой производительности.

После того, как были рассмотрены различные виды БД, приведем существующие аналоги предполагаемой разработки. На данный момент существует несколько общеизвестных баз данных для отделений неотложной медицинской помощи:

- федеральная медицинская информационная система (ФМИС): ФМИС является единым информационным ресурсом для системы здравоохранения в России. Она объединяет различные базы данных и модули, включая модуль неотложной медицинской помощи. В рамках этой системы регистрируются и хранятся данные о поступающих в отделения неотложной помощи пациентах. В ФМИС включены сведения о пациентах, результаты обследований и лечения, информация о медицинских событиях;
- единая автоматизированная информационная система "Амбулатория": эта система разработана для работы с амбулаторными картами пациентов и учета медицинской информации. Она может быть использована в поликлиниках и отделениях неотложной помощи. В рамках системы "Амбулатория" регистрируются пациенты, ведется электронная история болезни, хранятся результаты обследований и лечения. Система также позволяет врачам планировать визиты, назначать лекарства и процедуры, а также отслеживать текущее состояние пациента;
- система учета вызовов скорой медицинской помощи (СУВ): эта система предназначена для приема и учета вызовов скорой помощи. Она используется диспетчерскими службами скорой помощи для регистрации информации озывающем лице, месте происшествия, описании ситуации и оценке состояния пациента. СУВ позволяет оперативно отправлять бригады скорой помощи на место происшествия и отслеживать их движение в реальном времени. В системе также регистрируются данные о времени прибытия бригады, оказанной помощи и транспортировке

- пациента;
- информационная система "Регистр оказания скорой медицинской помощи"(РОСМЕД): эта система используется для сбора и анализа данных о оказании скорой медицинской помощи в России. РОСМЕД предназначена для регистрации информации о вызовах скорой помощи, диагнозах, примененных медицинских манипуляциях и результатах оказания помощи. В систему вносятся данные о каждом вызове скорой помощи, включая информацию о времени вызова, причине вызова, медицинских мерах, принятых бригадой скорой помощи, а также о результате лечения или транспортировке пациента.

Эти базы данных - ФМИС, система "Амбулатория СУВ и РОСМЕД - представляют собой информационные системы, используемые в различных аспектах медицинской помощи в России. Они помогают в сборе, хранении и управлении медицинской информацией, что способствует более эффективному и качественному оказанию неотложной медицинской помощи пациентам.

Все существующие БД хорошо выполняют свой спектр функций, но ни одна из рассмотренных систем не дает автоматизации и упрощения работы медицинского работника непосредственно во время вызова. Работнику все равно приходится работать с бумажными носителями и заполнять все вручную, а лишь потом происходит оцифровка всех собранных данных о пациенте.

Наша же задача стоит в том, чтобы организовать весь спектр проводимых манипуляций с информацией от момента вызова, до момента возвращения бригады в ОНМП в электронном виде для ускорения работ и минимизации ошибок при заполнении медицинских карт, выставлении диагноза и оказания требуемой медицинской помощи.

1.3 Назначение и возможности базы данных

В предметной области системы заполнения электронных медицинских карт для отделения неотложной медицинской помощи, основные объекты и свойства, которые следует рассмотреть, могут включать:

- пациенты: информация о каждом пациенте, включая персональные данные (имя, дата рождения, пол и контактная информация),

- медицинскую историю, диагнозы, принятые мероприятия, результаты обследований и лекарственное лечение;
- медицинские работники: данные о врачах, медсестрах и других медицинских специалистах, включая их идентификационные данные, специализацию, график работы и доступные привилегии;
 - медицинские процедуры: информация о проводимых процедурах, включая коды процедур, описания, стоимость, требуемое оборудование и прочие детали;
 - отделения: данные о различных отделениях неотложной медицинской помощи в больнице, их назначение, доступный персонал и оборудование;
 - ресурсы: информация о доступных ресурсах, таких как медицинское оборудование, лекарства, материалы и другие необходимые средства;
 - расписание: график работы медицинского персонала и расписание доступности отделений и ресурсов;
 - системы управления и интеграция: необходимо учесть возможность интеграции с другими системами, такими как системы управления ресурсами и планирования процессов, чтобы обеспечить эффективное взаимодействие и координацию деятельности медицинских учреждений.

Для лучшего понимания предметной области, рассмотрим конкретный пример - разработку базы данных для отделения неотложной медицинской помощи.

Медицинские работники оказывают медицинскую помощь пациентам с острыми заболеваниями и травмами, которые требуют немедленного вмешательства, также необходимо быстро и точно определить диагноз, назначить лечение и принять меры по сохранению жизни пациента.

Одной из основных задач базы данных для отделения неотложной медицинской помощи является хранение и обработка медицинских данных пациентов, включая информацию о симптомах, диагнозах, назначенных лекарствах, процедурах и т.д. Кроме того, необходимо учитывать, что пациенты могут обращаться за медицинской помощью в нескольких отделениях неотложной медицинской помощи, поэтому база данных должна позволять обмениваться информацией между различными медицинскими учреждениями.

Для решения этих задач можно использовать реляционную базу данных, в которой каждый пациент будет представлен в виде отдельной записи в таблице, содержащей данные о пациенте, диагнозах, лекарствах и т.д. Ключами в таблицах могут быть номера пациента, номера записи и т.д. Это позволит производить выборку данных о конкретном пациенте, обращаться к истории его болезни, проводить анализ данных и выявлять тенденции в заболеваемости и лечении.

Однако, реляционная модель может столкнуться с проблемами при работе с полуструктурными и неструктурными данными, такими как медицинские изображения, видео и тексты медицинских записей. Для работы с этими данными может использоваться нереляционная база данных, такая как MongoDB. В MongoDB данные могут быть храниться в более гибкой форме, используя форматы, такие как JSON и BSON. MongoDB также позволяет хранить и обрабатывать файлы в формате BLOB (binary large object), что делает его идеальным инструментом для хранения медицинских изображений и других неструктурных данных.

Еще одной важной задачей при разработке базы данных для отделения неотложной медицинской помощи является обеспечение безопасности хранения и доступа к медицинским данным. Для этого может использоваться различные меры, такие как шифрование данных, авторизация пользователей и аудит доступа.

Также при проектировании базы данных для отделения неотложной медицинской помощи необходимо учитывать требования к ее масштабируемости, отказоустойчивости и производительности. В случае большого количества пациентов и медицинских записей может потребоваться использование кластерной архитектуры, репликации данных и других технологий, позволяющих обеспечить высокую доступность и производительность базы данных. Также необходимо обрабатывать большие объемы данных и поддерживать быстрый доступ к ним [3].

На сегодняшний день существует множество различных систем управления базами данных, которые могут быть использованы для разработки базы данных для отделения неотложной медицинской помощи, включая MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server, MongoDB и др. Выбор конкретной системы управления базами данных зависит от требований к производительности, масштабируемости, отказоустойчивости и других

факторов.

Анализируя все вышеперечисленные требования к организации базы данных, можно сделать вывод, что основным субъектом данной базы данных является врач, который будет осуществлять выезд и непосредственное оказание всех необходимых медицинских услуг.

Врач может иметь доступ к данным о препаратах, находящихся в использовании у данного наряда, он может осуществить поиск по названию лекарственного средства для оперативного предоставления ответа по запросу клиента.

Основные реализуемые функции:

- аутентификация и авторизация;
- разграничение ролей пользователей;
- добавление и удаление данных;
- поиск данных по нескольким критериям;
- вывод вспомогательных информационных таблиц.

Выводы по разделу

Проектирование базы данных для системы заполнения электронных медицинских карт является сложным процессом, требующим тщательного анализа предметной области, определения требований и границ проекта, выбора подходящей модели данных и СУБД, обеспечения безопасности данных, обучения пользователей и поддержки системы. Это комплексный процесс, который должен выполняться с участием экспертов в области медицины и баз данных для достижения оптимального результата. Результатом успешной разработки будет эффективная система управления медицинской информацией, способствующая повышению качества медицинской помощи, оптимизации процессов и улучшению результатов лечения пациентов.

В конечном итоге, разработка базы данных для отделения неотложной медицинской помощи является сложной задачей, которая требует учета множества факторов, таких как безопасность, масштабируемость, отказоустойчивость и производительность. Решение этих задач требует использование различных технологий и систем управления базами данных, а также тщательного анализа требований и потребностей пользователей.

2 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА

2.1 Выбор программных средств для реализации базы данных

Выбор программных средств играет большое значение для проектируемой базы данных, поскольку определенные на этом этапе программные продукты, средства разработки и технологии непосредственным образом окажут влияние на распространение разработанной продукта: чем более распространенными будут используемые при разработке и требуемые для функционирования базы данных технологии, тем шире круг потенциальных потребителей, имеющих возможность ей воспользоваться. Кроме того, выбор среды разработки непосредственно влияет на стоимость и простоту процесса разработки в целом.

База данных «оптр» является реляционной. Реляционная база данных представляет собой набор таблиц, однако достаточно часто в состав базы данных входят и другие элементы, позволяющие дополнительно влиять на организацию и структуру данных в соответствии с определенным набором требований.

Создание и развитие динамических веб-страниц требует использования различных технологий. Разработка динамических веб-страниц включает три основных компонента: веб-сервер, язык программирования сценариев, исполняемых на стороне сервера, и базу данных.

Язык структурированных запросов (Structured Query Language, SQL) – самый распространенный язык, предназначенный для записи, извлечения, обновления и удаления информации в системах управления реляционными базами данных.

PostgreSQL (полное название «PostgreSQL: The world's most advanced open source relational database») - это мощная объектно-реляционная система управления базами данных (СУБД), которая поддерживает SQL-запросы и соответствует многим стандартам ANSI SQL. PostgreSQL является свободным и открытым программным обеспечением, доступным для использования и модификации бесплатно.

PostgreSQL обладает рядом преимуществ по сравнению с другими СУБД [4; 5] :

- надежность и целостность данных: PostgreSQL имеет высокую

степень надежности и обеспечивает высокую степень целостности данных благодаря поддержке транзакций, атомарности, согласованности и изоляции (ACID);

- безопасность: PostgreSQL обеспечивает множество функций безопасности, включая возможность управления правами доступа на уровне таблиц, столбцов, функций и процедур, проверку подлинности и защиту от SQL-инъекций;
- масштабируемость: PostgreSQL обладает высокой степенью масштабируемости и может обрабатывать большие объемы данных. Он поддерживает репликацию данных и многоуровневую архитектуру серверов;
- поддержка SQL: PostgreSQL поддерживает стандарт SQL и соответствует многим стандартам ANSI SQL, что облегчает разработку и поддержку приложений;
- расширяемость: PostgreSQL поддерживает расширения, которые позволяют разработчикам создавать свои собственные функции, типы данных и языки программирования, расширяя тем самым функциональность базы данных;
- мощный функционал: PostgreSQL имеет широкий функционал, включая поддержку полнотекстового поиска, гео-пространственных запросов, JSON- и XML-обработки, а также многопоточности;
- открытость: PostgreSQL является свободным и открытым программным обеспечением, доступным для использования и модификации бесплатно, что обеспечивает независимость от поставщика и поддержку со стороны большого сообщества разработчиков и пользователей.

2.2 Проектирование базы данных

Проектирование базы данных для любой автоматизированной системы разделяется на следующие этапы [6]:

- определение требований к базе данных;
- концептуальное проектирование базы данных;
- выбор средств реализации базы данных;
- логическое проектирование;
- физическое проектирование.

Объекты, которые хранятся в базе данных, имеют некую логическую структуру, то есть описываются некоторой моделью представления данных. К числу классических моделей относят следующие: иерархическая, сетевая и реляционная.

Иерархическая модель представляет собой упорядоченную совокупность данных, организованных в виде дерева, где каждый узел содержит записи данных. Тип «дерево» является составным, состоящим из подтипов (поддеревьев), которые также являются типом "дерево". Каждый тип "дерево" состоит из одного корневого типа и упорядоченного набора подчиненных типов, устанавливая связь «предок-потомок».

Сетевая модель отображает различные взаимосвязи данных в произвольном графе и обобщает иерархическую модель данных. Она использует два типа данных: записи и связи. Связь определяется между двумя типами записей - предком и потомком. В отличие от иерархической модели, где потомок имеет только одного предка, здесь потомок может иметь произвольное число предков.

Реляционная модель основывается на понятии отношений и представляет собой множество элементов, называемых кортежами. Отношение часто представляется в виде двумерной таблицы. Каждая строка таблицы имеет одинаковую структуру и состоит из полей. Строкам соответствуют кортежи, а столбцам - атрибуты. Реляционная модель удобна для описания простых связей между объектами, где каждому объекту соответствует строка таблицы. Если требуется описать более сложные логические структуры данных, часто используют связывание нескольких таблиц.

Для реализации данной медицинской информационной системы будем использовать реляционную модель.

Концептуальная модель базы данных - это наглядная диаграмма, которая использует стандартные обозначения и детально показывает связи между объектами и их характеристиками. Концептуальная модель создается для последующего проектирования базы данных и ее преобразования, например, в реляционную базу данных. В концептуальной модели визуально отображаются связи между объектами данных и их характеристиками [7].

ER-модель – модель данных, позволяющая описывать концептуальные схемы на основе диаграмм сущность-связь (ER-диаграмм).

Модель базы данных описывают с помощью одной или нескольких ER-диаграмм, содержащих сущности, атрибуты и связи.

Сущность определяется как объект, событие или концепция, информация о котором должна сохраняться. Сущности имеют наименование, несущее четкое смысловое значение. Каждый экземпляр сущности на диаграмме уникален.

Атрибут хранит информацию об определенном свойстве сущности и имеет четкое смысловое значение. Атрибут или группа атрибутов, которые однозначно идентифицируют экземпляры сущности, называются первичным ключом (англ. primary key).

Связь описывает логическое соотношение между сущностями. Связь сущности с другими сущностями определяет ее тип: различают два типа сущностей – зависимые и независимые.

Можно установить следующие связи между сущностями: идентифицирующая связь «Один–ко–Многим», связь «Многие–ко–Многим», неидентифицирующая связь «Один–ко–Многим» и связь «Один–к–Одному».

Связь «Многие–ко–Многим» существует только на логическом уровне. При переходе на физический уровень это отношение должно быть преобразовано за счет добавления новой зависимой сущности, связанной идентифицирующими связями «Один–ко–Многим» с сущностями, находящимися в исходном отношении.

Сущности, имеющие связь «Один–к–Одному», можно объединить в одну. При физической реализации базы данных две таблицы могут использоваться вместо одной по соображениям конфиденциальности, для удобства, для экономии дискового пространства, из семантических соображений.

Идентифицирующая связь устанавливается между независимой и зависимой сущностями, при этом зависимая сущность не может существовать самостоятельно - экземпляр зависимой сущности определяется только через отношение к сущности, которая его идентифицирует. При установлении идентифицирующей связи атрибуты первичного ключа родительской сущности автоматически переносятся в состав ключевых атрибутов дочерней сущности. В дочерней сущности новые атрибуты помечаются как внешний ключ. В случае неидентифицирующей связи внешний ключ не входит в состав первичного ключа дочерней сущности.

2.3 Проектирование логических моделей данных

Проектирование модели данных состоит из двух уровней представления данных - логического и физического. Такое разделение на модели позволяет разделять задачу на более мелкие элементы.

Логический уровень – это некоторая абстрактная модель данных, позволяющая описать исследуемый объект, подчеркивая в нем необходимые свойства. Преимущество этого уровня заключается в том, что он является универсальным, поэтому реализовав его, можно создать БД, используя всевозможные для этого инструменты.

Различают три уровня логической модели, отличающихся по глубине представления информации о данных:

- диаграмма сущность-связь (Entity Relationship Diagram, ERD);
- модель данных, основанная на ключах (Key Based model, KB);
- полная атрибутивная модель (Fully Attributed model, FA).

Диаграмма сущность-связь – этот тип логической модели используется при первоначальной разработке БД. Он позволяет описать основные объекты или процессы, необходимые для реализации в БД. Для этого вводятся 3 понятия: сущность, связь, атрибут. Сущность – это объект, находящийся в БД. Используя сущности, можно описать основные таблицы в будущей БД. Связь – это соединение или введение некоторого взаимодействия между сущностями. Если интерпретировать связь в БД, то связь выступает в роли внешних ключей. Атрибут – это ключевые элементы сущности, необходимые для его описания. В БД атрибут можно сравнивать со столбцами, хранящимися в таблице. Этот тип моделирования позволяет опускать описание атрибутов, чтобы не загромождать диаграмму.

Модель данных, основанная на ключах – это дополненная ER-диаграмма. Ее главное отличие от первой: обязательное наличие некоторых атрибутов. Этими атрибутами являются первичные ключи, необходимые для соблюдения уникальности данных, и внешние ключи, обеспечивающие целостность данных. Стоит отметить, что внешние ключи никак не изображаются, они лишь являются следствием связей между сущностями. Причем внешний ключ находится у слабой сущности, а сам этот ключ ссылается на первичный ключ сильной сущности.

Первичные ключи принято обозначать как РК – Primary Key,

графически они обозначаются двумя подчеркнутыми линиями, также допускается использование жирного шрифта. Внешние ключи принято обозначать как FK – Foreign Key, графически обозначаются одной подчеркнутой линией, допускается использование курсивного шрифта.

Полная атрибутивная модель - самая детализированная модель на логическом уровне. В ней исключается лишняя или дублирующая информация – этот процесс называется нормализацией БД. Помимо минимизации избыточной информации, в этой модели представляются все существующие сущности, связи и атрибуты.

Для построения полной атрибутивной модели используется IDEF1X нотация. В ней основными объектами также являются сущности, связи и атрибуты. Сущности называются в единственном числе и имеют четкое смысловое значение. Атрибуты должны именоваться в единственном числе и иметь четкое смысловое значение. Каждая связь должна именоваться глаголом или глагольной фразой, причем глагол ставится от 3 лица.

2.4 Нормализация базы данных

Для того, чтобы оптимизировать модель и избавиться от избыточных данных, проводят нормализацию данных. Нормализация – разбиение таблицы на две или более, обладающие лучшими свойствами при добавлении, изменении и удалении данных. Нормализация осуществляется с целью оптимизации объема БД и быстродействия запросов. Всего существует пять нормальных форм, но реально, на практике, используются лишь третья, а именно база данных в третьей нормальной форме (ЗНФ). Процесс нормализации отношений осуществляется пошагово и заключается в последовательном переводе отношения от первой нормальной формы к нормальным формам более высокого порядка. При этом каждая следующая нормальная форма сохраняет все свойства предыдущих.

Нужно проверить, находится ли полученная схема отношений в третьей нормальной форме (ЗНФ). Если схема отношений не находится в ЗНФ, то ее нужно нормализовать для минимизации избыточности данных и устранения потенциальной противоречивости данных.

Отношение находится в первой нормальной форме (1НФ), если значения всех атрибутов атомарные, то есть значение атрибута не должно быть множеством или повторяющейся группой.

Отношение находится во второй нормальной форме (2НФ), если оно находится в 1НФ , и нет частичной функциональной зависимости неключевых атрибутов от ключа (зависимость не ключевых атрибутов от части ключа). Из схемы отношений видно, что ни один из неключевых атрибутов функционально полно не зависит от ключа, следовательно, схема отношений находится в 2НФ .

Отношение находится в 3НФ , если оно находится во 2НФ , и отсутствуют транзитивные зависимости неключевых атрибутов от ключа. Между атрибутами А и С есть транзитивная зависимость, если выполняется совокупность условий: если хотя бы одно из условий не выполняется, то транзитивной зависимости между атрибутами А, В, С нет. Причем атрибуты А, В, С могут быть составными [8].

Анализируя атрибуты, можно сделать вывод, что транзитивная зависимость отсутствует, то есть отношение находится в 3НФ . Следовательно, все схемы отношений являются окончательными схемами отношений.

2.5 Проектирование физических моделей данных

В отличие от логической модели, физическая модель строится в зависимости от выбранной СУБД. Также этот вид модели позволяет напрямую создавать команды для системного каталога СУБД. Главное отличие физической от логической модели состоит в том, что в первой уделяется внимание на типы объектов. Так, атрибуты могут иметь разные форматы данных: например, числовой или строчный. Помимо формата данных, физическая модель допускает использование физических объектов СУБД: например, процедуры.

Различают два уровня физической модели:

- трансформационная модель (Transformation Model);
- модель СУБД (DBMS Model).

Трансформационная модель содержит информацию для реализации отдельного проекта, который может быть частью общей ИС и описывать подмножество предметной области. Трансформационная модель позволяет проектировщикам и администраторам БД лучше представлять, какие объекты БД хранятся в словаре данных, и проверить, насколько физическая модель данных удовлетворяет требованиям к ИС.

Модель СУБД является точным описанием того, как выглядит будущая

БД в системном каталоге СУБД. Именно эта модель физического уровня будет использоваться в данной работе, потому что она имеет практическое влияние.

Самым главным объектом в БД является таблица. Если интерпретировать физическую модель через логическую, то получим такой вывод, что таблицы являются сущностями, столбцы - атрибутами, а связи образуются через внешние ключи. Таблица содержит столбцы, именно поэтому им стоит уделить достойное внимание.

Первое свойство столбца, про которое стоит сказать, может ли оно принимать нулевые значения, в случае если может, нужно писать этому столбцу предложение «NULL», иначе «NOT NULL». Причем первичные ключи не могут быть нулевыми значениями, также не рекомендуется использовать «NULL» для внешних ключей. Второе свойство столбца есть сам формат данных, для целочисленных (причем 0 в старшем бите, не являющимся значимым) стоит использовать «INTEGER», для строковых значений можно использовать «VARCHAR(n)», а для использования даты применяется «DATE», а если нужна дата с временем, то «DATETIME» [9].

2.6 Готовые решения

Проанализировав программные средства для реализации баз данных, рассмотрев все основные принципы построения столбцов, создания первичных и внешних ключей и нормализацию данных, приведем ER-диаграммы основных структур [10].

Логично предположить, что для начала нам нужно будет создать аккаунт и пройти аутентификацию в приложении, пройдя верификацию по логину и паролю. Здесь возможно два исхода:

- верификация прошла успешно;
- отказано в доступе.

Как уже говорилось ранее, у нас возможны некоторые роли:

- администратор;
- пользователь.

В зависимости от этого у нас будут формироваться данные входа для разных ролей. Соответственно для этого нам нужно сделать структуру таблиц под эти требования. На рисунке 1 показана ER-диаграмма организации структуры для хранения данных о пользователе с определенными правами, ролями и группами.

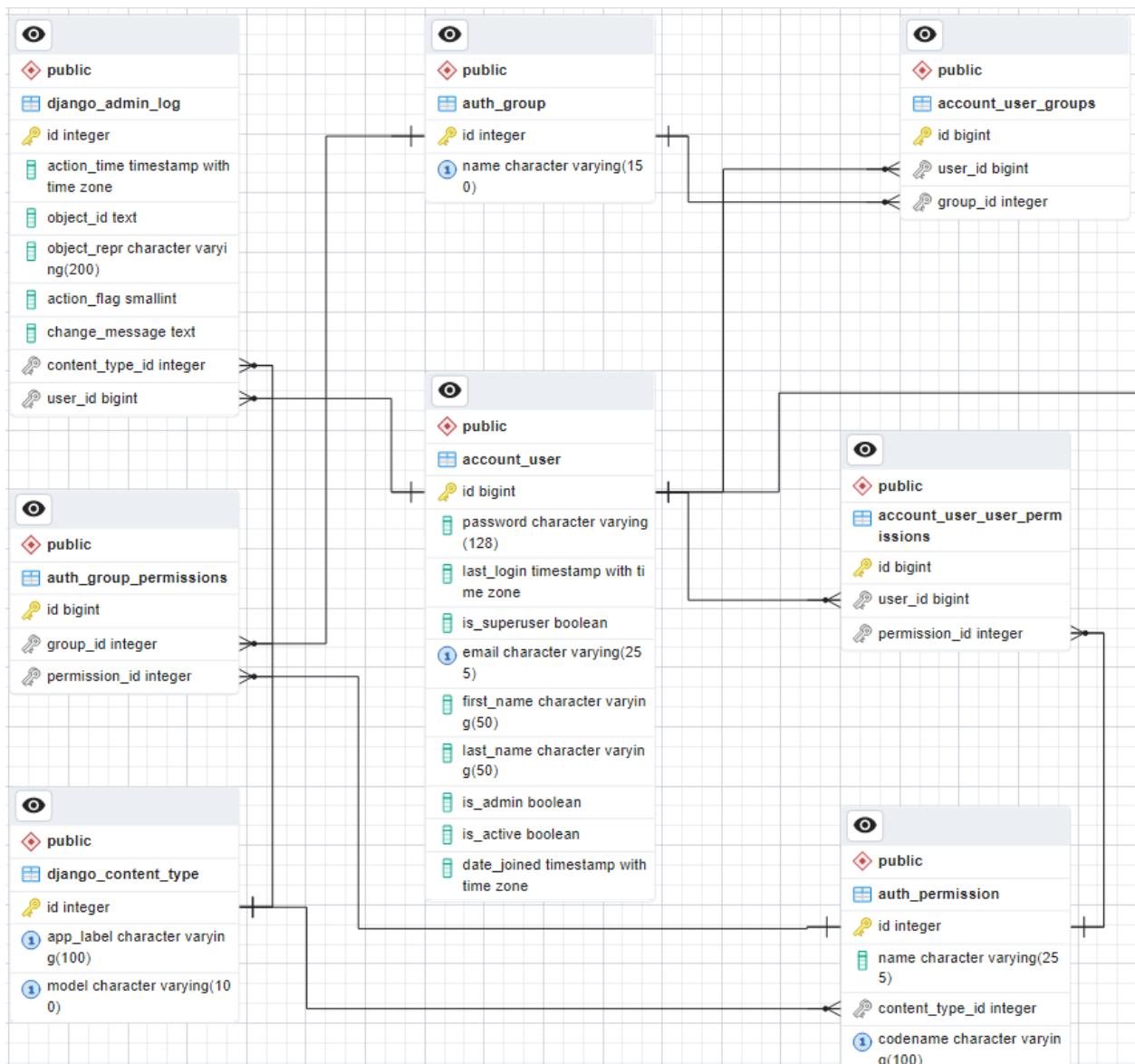


Рисунок 1 – Данные пользователя с определенными правами, ролями и группами

Как можно заметить, на рисунке 1 видна еще одна связь, которая уходит за пределы изображения. Это не случайно, сейчас объясним этот момент.

На рисунке 2 показана ER-диаграмма организации структуры хранения данных для связи нашей БД с взаимодействиями на стороне back-end'a. Есть таблица, которая заполняется при первичной регистрации пользователя и служит для информации на стороне back-end'a. Кроме того, есть таблица пользователя, которая нужна только в БД и служит непосредственно для связи пользователя с медицинскими картами пациентов, которые заполняет медицинский работник при выезде.

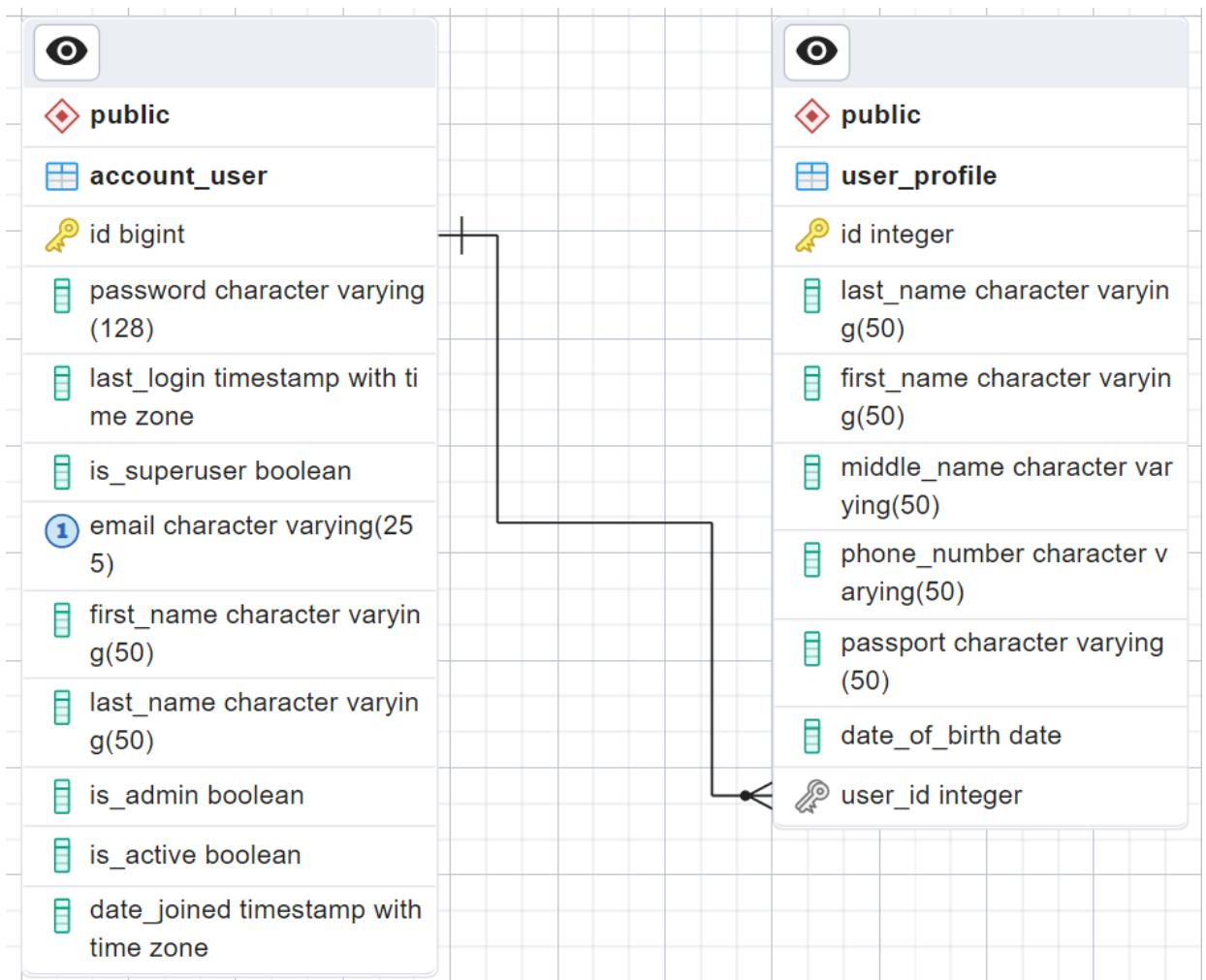


Рисунок 2 – Связи взаимодействия

На рисунке 2 была схематично показана связь двух таблиц. Теперь приведем описание таблицы `user_profile` и объясним ее назначение в нашей БД. На рисунке 3 показана ER-диаграмма организации структуры хранения данных для пользователей нашего приложения.

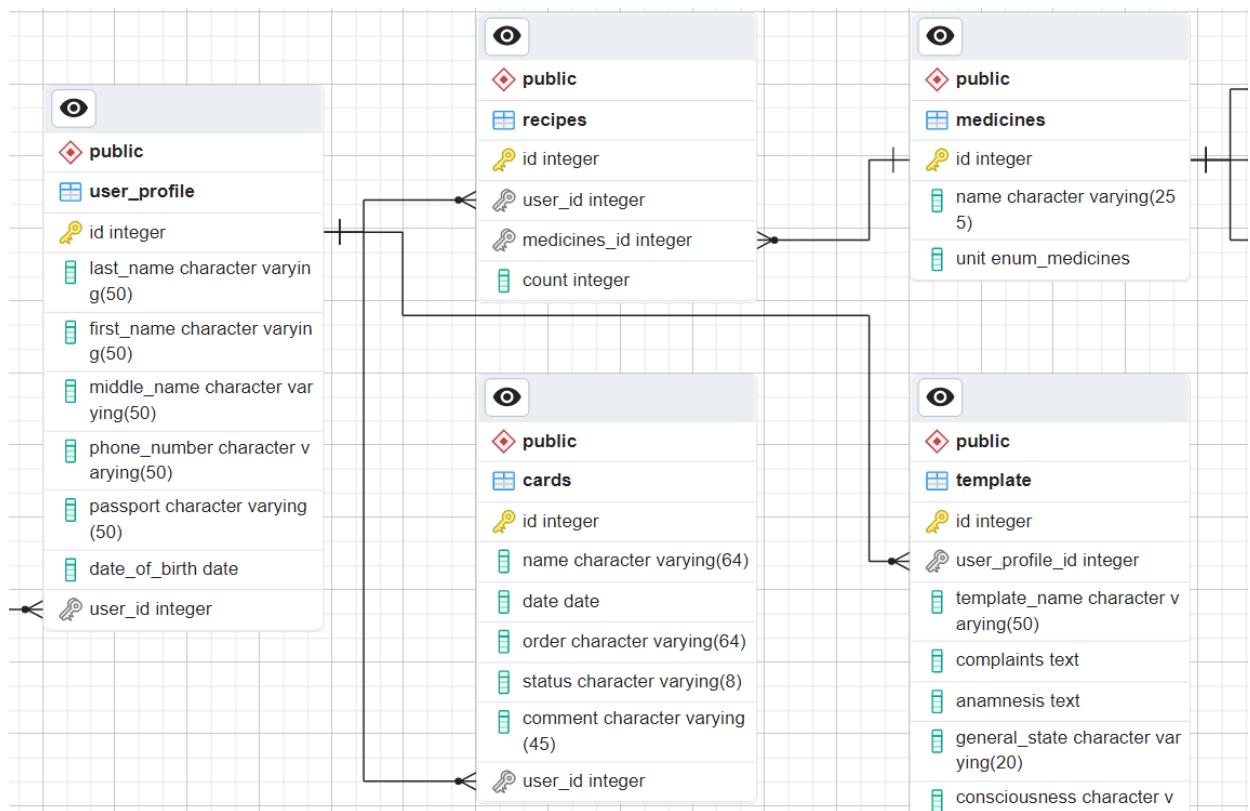


Рисунок 3 – Пользователь

Как можно заметить, в таблице *medicines* есть еще три связи, которые не попали на данный рисунок. Эти связи будут продемонстрированы на рисунке 7.

На рисунке 4 показана ER-диаграмма организации структуры хранения данных полного взаимодействия с момента регистрации и аутентификации пользователя в приложении до связи определенного медицинского работника с шаблоном для заполнения медицинской карты при выезде к пациенту.

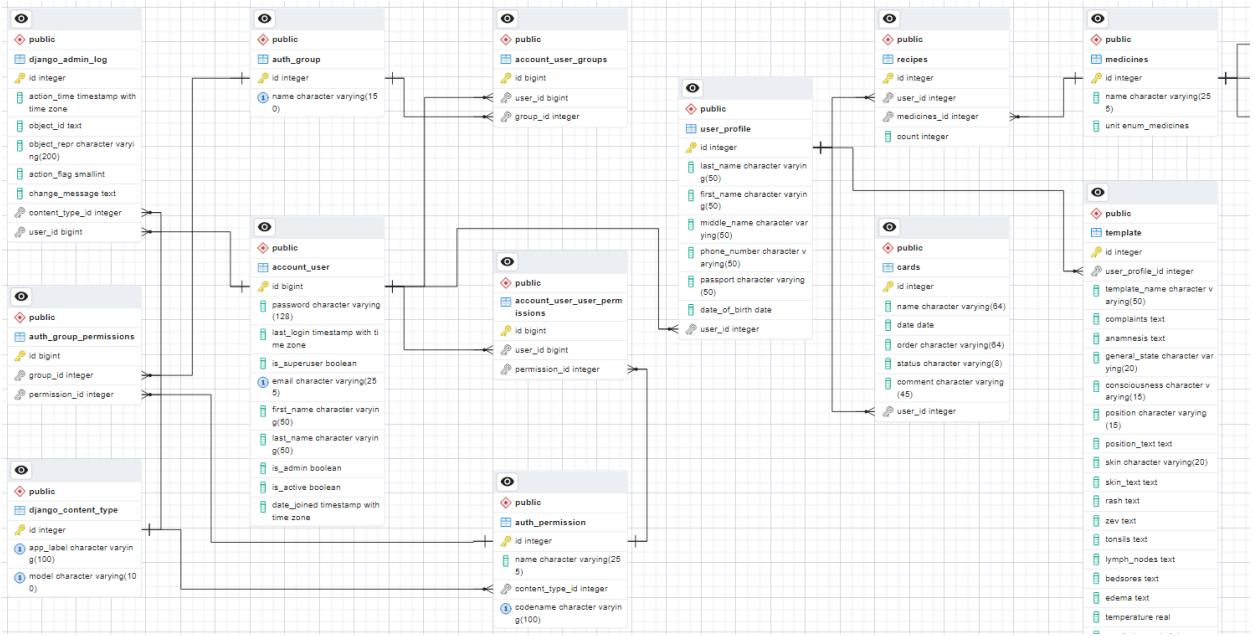


Рисунок 4 – Полная структура взаимодействия

На рисунке 5 показана ER-диаграмма организации структуры хранения данных для диагнозов. Каждый диагноз относится к определённому направлению в медицине - так называемый тег. Каждый диагноз имеет свой определенный код МКБ. У диагноза есть свой определенный перечень оказания объема необходимой медицинской помощи и тактика выполнения действий. Кроме того, диагноз имеет свои так называемые формы(поддиагнозы) и относящийся уже к ним объем необходимой медицинской помощи.

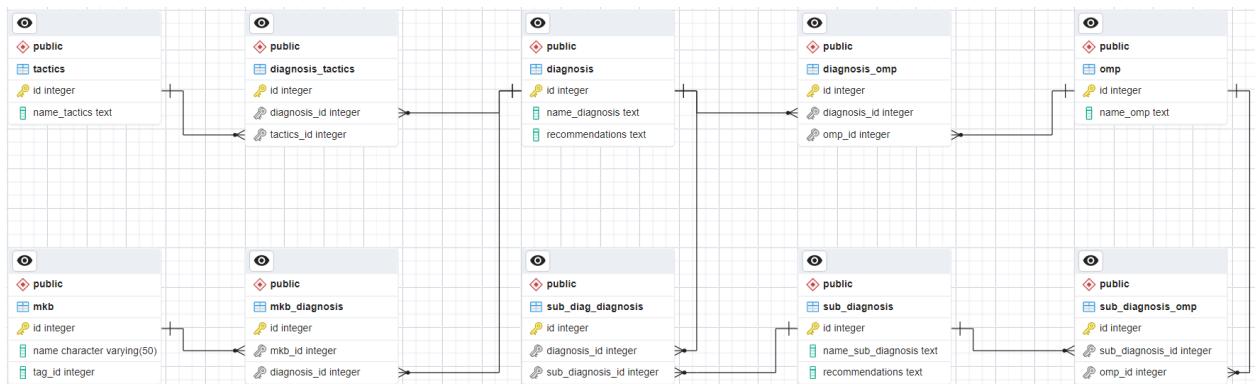


Рисунок 5 – Диагнозы

На рисунке 6 показана ER-диаграмма организации структуры хранения данных для заболеваний. Каждое заболевание относится к определённой категории в медицине - так называемый тег. У заболеваний есть свои

определенные симптомы. Кроме того, заболевание имеет свой собственные формы и относящиеся уже к этой форме симптомы.

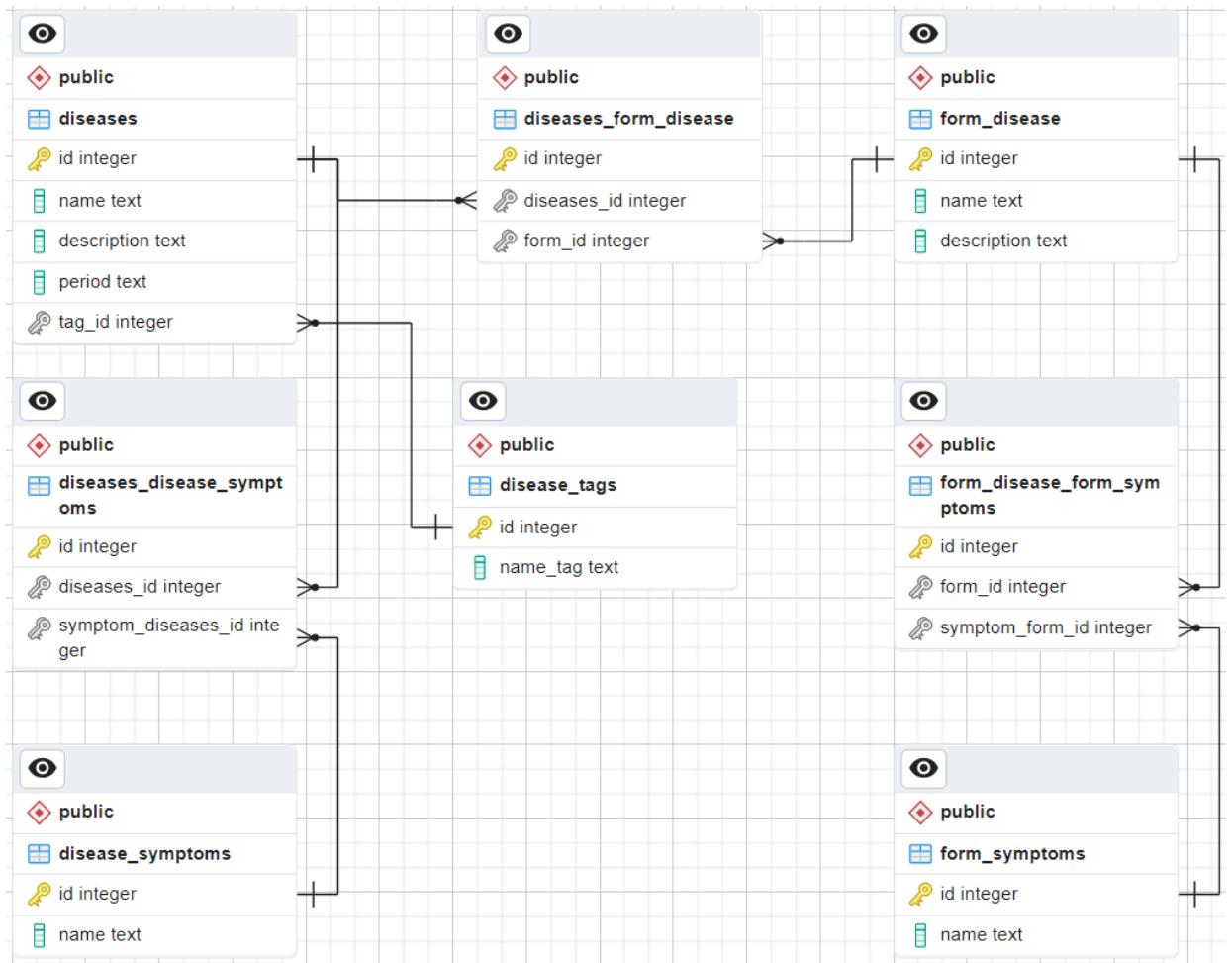


Рисунок 6 – Заболевания

На рисунке 7 показана ER-диаграмма организации структуры хранения данных для медикаментов. Каждый препарат имеет определенные противопоказания, взрослую и детскую дозировки, которые, в свою очередь зависят от конкретного диагноза.

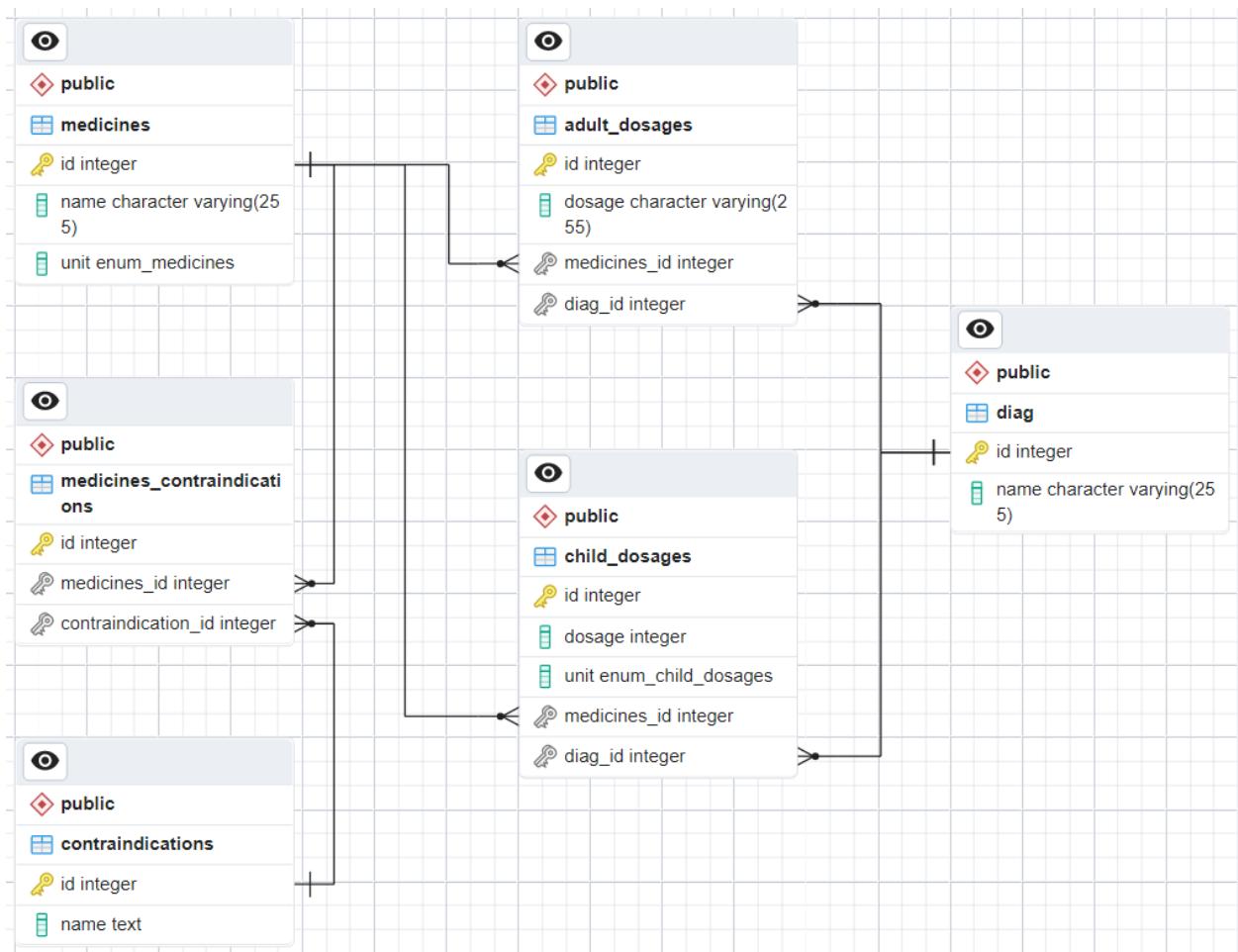


Рисунок 7 – Медикаменты

Конечно, это не все таблицы, требуемые для функционала целого приложения. Медицинскому работнику иногда требуется какая-то незначительная вспомогательная информация и какой-то сложной организационной структуры эти таблицы не требуют. Вывод этих таблиц можно будет увидеть при выполнении запросов.

Таким образом, были объявлены основные таблицы БД, были рассмотрены столбцы, а именно проанализированы типы данных, а также определены ограничения для внесения конкретных значений. Этого достаточно, чтобы между таблицами сохранялась целостность данных, а вставка значений была корректной [11; 12; 13].

Важно понимать, что перед внедрением базы данных необходимо провести тестирование и проверку ее функциональности, производительности и безопасности. Это поможет выявить потенциальные проблемы и ошибки, которые могут повлиять на работу системы. Регулярное тестирование после внедрения также является важной частью обслуживания базы данных.

Выводы по разделу

В этом разделе разбирались теоретические сведения, связанные с созданием БД. С помощью анализа организации и метода моделирования мы смогли решить задачи, связанные с определением предметной области ОНПМ и разработкой логических и физических моделей БД. На основе этих моделей были созданы таблицы в PostgreSQL, а для графического представления использовалось такое ПО как pgAdmin. Также была выполнена практическая задача, связанная с формированием системного каталога структуры БД.

Важно отметить, что разработка базы данных не является заключительным шагом. Необходимо предусмотреть систему поддержки и обслуживания, которая будет обеспечивать регулярные обновления базы данных, исправление ошибок, мониторинг производительности и резервное копирование данных. Регулярное техническое обслуживание поможет сохранить работоспособность и безопасность базы данных на протяжении всего ее существования [14].

3 ЗАЩИТА БАЗЫ ДАННЫХ ОТ НЕСАНКЦИОНИРОВАННОГО ДОСТУПА

3.1 Основные угрозы и уязвимости

Основные угрозы, связанные с безопасностью веб-приложений, включают:

- атаки на клиентов;
- утечка важных данных;
- НСД к приложению;
- НСД к функциональности или контенту;
- раскрытие конфигурационной информации;
- отказ в обслуживании;
- атаки на ресурсы;
- выполнение команд ОС на сервере.

На рисунке 8 представлена доля угроз в процентном соотношении:

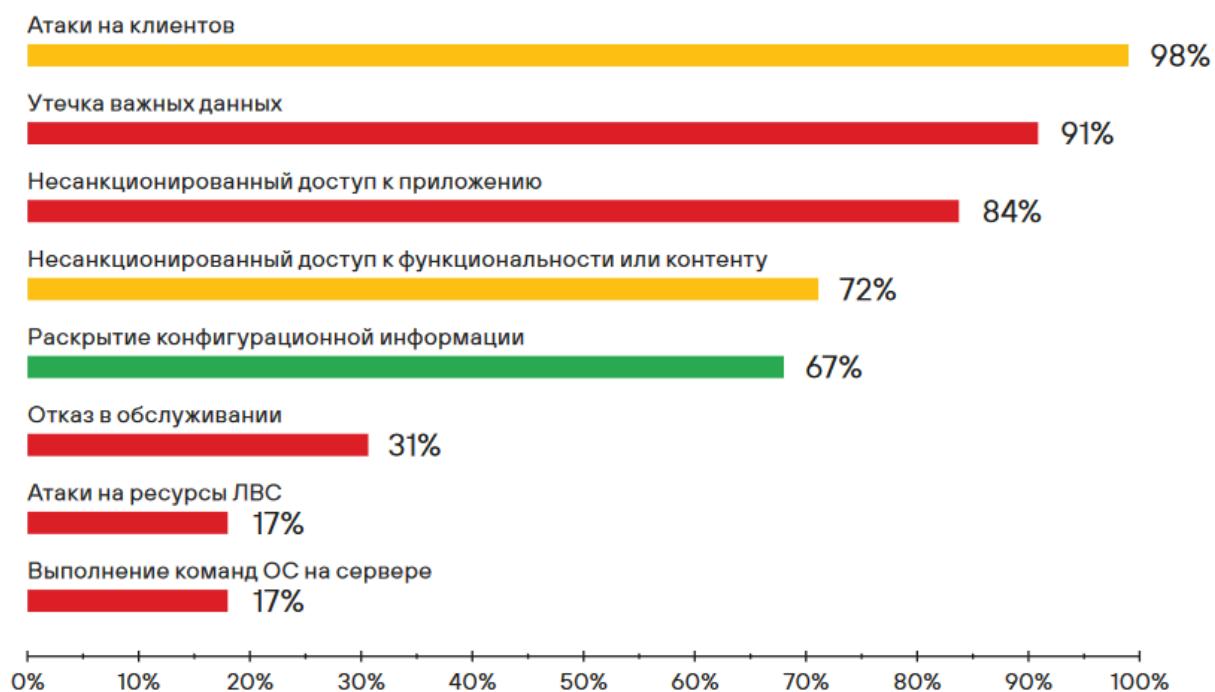


Рисунок 8 – Распространённые угрозы веб-приложений

Как можно заметить, утечка важных данных занимает второе место среди распространенных угроз на веб-приложения. Именно поэтому важно продумать систему защиты и хранения данных в БД.

Начиная с 2017 года актуальность атак на клиента возросла и возглавляет

топ актуальных угроз. Атака вида – XSS (cross-site scripting) является каждой второй в списке атак. Сутью XSS уязвимости является возможность внедрения в веб-систему вредоносного кода и дальнейшего взаимодействия этого кода с серверами злоумышленника. Частным примером является SQL-инъекции.

SQL уязвимость основана на внедрении в запрос части произвольного SQL кода. Существует несколько способов применения SQL-инъекций использование числового входящего параметра и использование строкового входящего параметра.

Для предотвращения SQL-injection необходимо проводить правильную обработку пользовательского ввода, используя специальные методы для экранирования специальных символов и проверки вводимых данных. Например, можно использовать подготовленные запросы, которые разделяют SQL-код и данные, передаваемые в запросе, тем самым защищая приложение от выполнения злонамеренного кода.

Также важно следить за обновлением и патчингом используемых баз данных и их драйверов, чтобы избежать известных уязвимостей, которые могут быть использованы злоумышленниками для SQL-injection.

В целом, защита от SQL-инъекций требует аккуратности и внимательности в проектировании и разработке веб-приложений, а также постоянного мониторинга на наличие уязвимостей и проведения регулярных тестов на проникновение.

3.2 Конфиденциальность персональных данных

При разработке базы данных важно учитывать требования безопасности и конфиденциальности медицинской информации. Для этого могут быть приняты следующие меры:

- шифрование данных: чувствительная медицинская информация, такая как персональные данные пациентов и медицинская история, должна быть зашифрована при хранении и передаче. Использование сильных шифровальных алгоритмов поможет защитить данные от несанкционированного доступа [15];
- многофакторная аутентификация: для обеспечения безопасности доступа к базе данных, особенно для медицинского персонала, следует реализовать многофакторную аутентификацию. Это требует предоставления нескольких форм идентификации, таких как

пароль, биометрические данные или специальные токены, для подтверждения легитимности пользователя;

- управление доступом: реализация гибкой системы управления доступом поможет ограничить доступ к конфиденциальной информации только уполномоченному персоналу. Ролевая модель доступа может быть использована для определения прав доступа на основе ролей и ответственостей сотрудников;
- аудит и мониторинг: важно вести аудит и мониторинг доступа к базе данных для обнаружения и предотвращения несанкционированного доступа или неправомерной активности. Журналы доступа и системы мониторинга позволяют выявить подозрительную активность и принять соответствующие меры;
- соответствие нормам и законодательству: разработка базы данных должна соответствовать применимым нормам и законодательству в области защиты персональных данных. Это включает в себя соблюдение правил по сбору, хранению, использованию и передаче медицинской информации;
- восстановление после катастрофы и резервное копирование: необходимо реализовать надежный план восстановления после катастрофы и резервного копирования, чтобы обеспечить доступность и целостность медицинской базы данных. Регулярные резервные копии должны выполняться, чтобы защитить от потери данных в случае сбоев системы, стихийных бедствий или кибератак. Также важно периодически тестировать процесс восстановления, чтобы убедиться в возможности точного и эффективного восстановления данных.

Анализируя вышеперечисленные пункты, можно сделать вывод, что обеспечение конфиденциальности хранения персональных данных является одним из ключевых вопросов, связанных с использованием цифровых медицинских карт и баз данных [16].

Для обеспечения конфиденциальности данных необходимо проводить работу по защите информации и соблюдению законодательных требований. Организация должна предпринимать меры для защиты персональных данных пациентов, такие как использование паролей и шифрования данных. Кроме того, необходимо обучать медицинских работников основам информационной

безопасности и контролировать доступ к информации.

Конфиденциальность хранения персональных данных пациентов - это важный аспект использования цифровых медицинских карт и организации должны принимать соответствующие меры для обеспечения безопасности информации и соблюдения законодательных требований.

PostgreSQL поддерживает множество функций для обеспечения безопасности данных, включая конфиденциальность хранения персональных данных.

Одним из ключевых механизмов для обеспечения конфиденциальности данных в PostgreSQL является использование различных методов шифрования, включая шифрование данных в пути и в покое, а также использование SSL для защиты соединений.

PostgreSQL также обеспечивает механизмы авторизации и аутентификации, которые позволяют контролировать доступ к базе данных и ее объектам, таким как таблицы и представления. Например, администраторы могут назначать различные роли и права доступа к объектам базы данных, что позволяет управлять доступом к конфиденциальным данным.

PostgreSQL также поддерживает аудиторскую функциональность, которая позволяет записывать действия пользователей в базе данных, такие как входы и выходы из системы, выполнение запросов и изменение данных. Это позволяет отслеживать и анализировать действия пользователей, чтобы обеспечить безопасность данных и защитить их от несанкционированного доступа.

В целом, PostgreSQL - это надежная и безопасная реляционная база данных, которая обеспечивает множество механизмов для защиты конфиденциальности данных, включая персональные данные пациентов.

3.3 Использование имен пользователей, ролей и разрешений

PostgreSQL обладает базовым механизмом защиты, включающим в себя идентификацию, аутентификацию и авторизацию. Идентификация – проверка на то, существует ли данный субъект, желающий воспользоваться данным ресурсом. Следующим шагом после идентификации следует аутентификация – проверка подлинности субъекта, в основном для этого используется паролевый метод. Конечным шагом является авторизация – предоставление необходимых прав субъекту или отказ в доступе к нужным ресурсам. Для

реализации описанного механизма применяются учетные записи пользователей.

На рисунке 9 представлено создание учетных записей для работников скорой медицинской помощи, а именно:

- главный врач бригады - head_physician;
- врач - doctor;
- фельдшер - paramedic;
- медсестра - nurse.

```
Query History
1 CREATE USER head_physician
2 WITH ENCRYPTED PASSWORD 'qwerty123';
3
4 CREATE USER doctor
5 WITH ENCRYPTED PASSWORD 'qwerty123';
6
7 CREATE USER paramedic
8 WITH ENCRYPTED PASSWORD 'qwerty123';
9
10 CREATE USER nurse
11 WITH ENCRYPTED PASSWORD 'qwerty123';
```

Рисунок 9 – Создание учетных имен пользователей

Стоит остановиться на параметрах «CHECK_EXPIRATION» и «CHECK_POLICY», которые относятся к управлению паролями и их проверке при создании или изменении пользовательских данных. «CHECK_EXPIRATION» отвечает за то, есть ли у пароля срок истечения, а «CHECK_POLICY» определяет, должна ли использоваться пароловая политика. По умолчанию, проверка срока действия пароля и политики паролей включена в PostgreSQL, поэтому опции CHECK_EXPIRATION и CHECK_POLICY не нужно указывать.

Теперь нам нужно убедиться, что учетные имена пользователей и правда были созданы. Это продемонстрировано на рисунке 10.

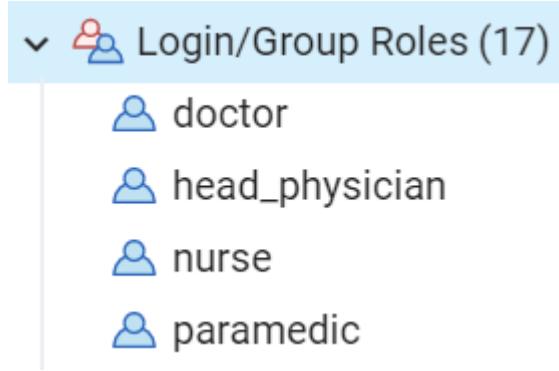


Рисунок 10 – Проверка создания учетных имен пользователей

Определив учетные записи пользователей, остается лишь предоставить им необходимые разрешения, позволяющие взаимодействовать с БД. Можно отдельно каждому пользователю прописывать разрешения, но в данном случае это нецелесообразно – потому что фельдшера и медсестры должны быть одинаковые права, а у главврача и врача они такие же, только к тем правам добавляются еще другие. Выходом из этой ситуации являются роли. Роль – это объект БД, позволяющий объединять пользователей в единую группу с целью эффективного администрирования. Создание ролей и присоединение пользователей в роли продемонстрировано на рисунке 11.

```
Query   Query History


---


1  CREATE ROLE junior;
2  GRANT junior TO paramedic, nurse;
3
4  CREATE ROLE senior;
5  GRANT senior TO head_physician, doctor;
```

Рисунок 11 – Создание ролей и присоединение пользователей в роли

Проверим, находятся ли учетные записи в определенных ролях. Факт такого наличия изображен на рисунке 12.

Group Role - junior

- General**
- Definition**
- Privileges**
- Membership** (selected)
- Parameters**
- Security**
- SQL**

Member of

User/Role	WITH ADMIN

Members

User/Role	WITH ADMIN
paramedic	<input type="checkbox"/>
nurse	<input type="checkbox"/>

Group Role - senior

- General**
- Definition**
- Privileges**
- Membership** (selected)
- Parameters**
- Security**
- SQL**

Member of

User/Role	WITH ADMIN

Members

User/Role	WITH ADMIN
head_physician	<input type="checkbox"/>
doctor	<input type="checkbox"/>

Рисунок 12 – Проверка на наличие пользователей в ролях

Создав роли для пользователей, нужно определить некоторые права, позволяющие манипулировать БД. Для этого в PostgreSQL предусмотрена возможность выдачи прав как отдельным пользователям, так и ролям.

Обозначим права, которые должны иметь роли. Роль «junior» должна иметь право на выборку всех таблиц существующей БД. Роль «senior» должна иметь право на выборку всех таблиц существующей БД, а также право на вставку, изменение и удаление значений всех таблиц, которые связаны с оказанием медицинской помощи и заполнением медкарты. На рисунке 13 представлено присвоение ролям необходимых прав.

Query Query History

```
1 GRANT SELECT ON ALL TABLES IN SCHEMA public TO junior;
2
3 GRANT SELECT ON ALL TABLES IN SCHEMA public TO senior;
4 GRANT INSERT ON ALL TABLES IN SCHEMA public TO senior;
5 GRANT UPDATE ON ALL TABLES IN SCHEMA public TO senior;
6 GRANT DELETE ON ALL TABLES IN SCHEMA public TO senior;
```

Рисунок 13 – Назначение прав ролям и пользователям

Стоит отметить еще одну возможность PostgreSQL. Для запрета на определенное право используется конструкция: «REVOKE action_name ON object_name FROM user_name/role_name;», при запрете права также может использоваться «CASCADE», это свойство наоборот запрещает некоторые права тем субъектам, которым оно выдавалось. Помимо выдачи и запрета прав есть третья возможность – отмена прав, она имеет следующий синтаксис: «REVOKE action_name ON object_name FROM user_name/role_name [CASCADE];», свойство «CASCADE» используется для отмены прав у тех субъектов, которым данный пользователь выдал. Стоит обратить внимание, что для использования команды REVOKE в PostgreSQL необходимо иметь соответствующие привилегии администратора базы данных.

pgAdmin также позволяет просмотреть права ролей у каждой таблицы. На рисунке 14 изображены предоставленные права к таблице «Медикаменты».

Privileges		
Grantee	Privileges	Grantor
junior	r	postgres
senior	awrd	postgres

Рисунок 14 – Проверка ролей на их возможные права в таблице

3.4 Шифрование базы данных

Вторым базовым механизмом защиты, реализующимся в PostgreSQL, от

НСД является шифрование БД. Шифрование – это преобразование исходной информацию в данные, которые теряют иной смысл, не зная секретного слова или алгоритма, преобразовывающие данные обратно в смысловую информацию.

Самый популярный способ зашифровать информацию, хранящуюся в БД – это прозрачное шифрование данных. Преимущество такого вида шифрования заключается в том, что конечные пользователи могут даже и не знать, что применяется защита информации.

Прозрачное шифрование, также известное как шифрование в режиме реального времени, представляет собой способ защиты данных, при котором шифрование и расшифровка осуществляются автоматически без участия пользователя. Этот метод основан на работе специального драйвера, который функционирует в фоновом режиме и следит за всеми операциями с данными. Его главная цель заключается в предотвращении атак, направленных на получение данных путем обхода операционной системы, например, через загрузку из другой ОС или использование альтернативных методов.

Перед тем, как перейти к шифрованию, следует, на всякий случай, сделать резервную копию БД. Для этого следует включить pgAgent.

Затем выбираем нужную БД, вызываем контекстное меню, в нем выбираем опцию «создать резервную копию».

Создав резервную копию БД, можно приступать к процессу шифрования. Этот процесс происходит в несколько этапов:

- создание главного ключа БД;
- создание сертификата;
- создание ключа шифрования;
- запуск процесса шифрования;
- проверка состояния шифрования.

Главный ключ БД – это ключ, который применяется для шифрования других ключей, использующихся для реализации шифрования в БД. Для его создания применяется следующая конструкция: «CREATE MASTER KEY ENCRYPTION BY PASSWORD 'password';».

Следующим шагом будет создание сертификата. Сертификат – это объект безопасности, имеющий подпись. Этот объект хранит в себе ключи шифрования. Для создания данного объекта применяется следующая конструкция: «CREATE CERTIFICATE name_db_and_cert WITH SUBJECT

'description';».

Последним подготовительным этапом перед началом шифрования БД является создание ключа шифрования. Именно этим ключом будет происходить шифрование БД, а сам ключ будет находиться в сертификате, созданном заранее. Для его создания требуется применять следующий шаблон: «CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES_128 ENCRYPTION BY SERVER CERTIFICATE cert_name;».

Следующий шаг является необязательным, но его желательно выполнять. При создании главного ключа, сертификата и ключа шифрования рекомендуется создание резервных копий этих объектов безопасности. Для создания резервной копии ключа шифрования можно использовать команду pg_dump, которая создаст резервную копию всей базы данных, включая ключи шифрования: «pg_dump dbname > backup_file_name;».

Все объекты безопасности проинициализированы, также для критических объектов были созданы резервные копии. Теперь можно приступить к шифрованию БД. Для начала данного процесса используется следующий пакет команд: «UPDATE table_name SET column_name = pgp_sym_encrypt(column_name, 'key_password');».

Осталось лишь проверить, работает ли ключ шифрования для БД. Для этого используются системные БД «pg_database_encryption». Нас интересует столбец is_encrypted, если шифрование завершилось успешно, то столбец принимает значение 1.

Но, сказать честно, нас мало интересует сквозное шифрование всей БД. Ведь сквозное шифрование всей базы данных может иметь несколько негативных аспектов:

- высокая нагрузка на производительность: Шифрование и дешифрование больших объемов данных может быть ресурсоемким процессом, что может замедлить операции чтения и записи в базе данных. Это особенно актуально для приложений с высокой нагрузкой или большим количеством одновременных пользователей;
- ограничения по поиску и сортировке: Шифрованные данные нельзя эффективно искать или сортировать без их предварительного расшифрования. Если в базе данных необходимо выполнять сложные операции поиска или агрегации данных, то сквозное

шифрование может существенно затруднить выполнение этих операций;

- управление ключами: Для сквозного шифрования необходимо использовать сильные шифровальные ключи, их генерацию, хранение и управление. Ключи должны быть доступны для расшифровки данных, что может представлять риск безопасности, особенно если ключи утекают или попадают в неправильные руки;
- затрудненная администрация: Сквозное шифрование требует дополнительного управления и конфигурации в базе данных и приложениях. Это может усложнить администрирование базы данных и требовать дополнительных усилий для поддержки и масштабирования системы;
- ограниченные возможности обработки данных: Шифрование данных делает их недоступными для анализа, обработки и использования в алгоритмах машинного обучения или других высокоуровневых операциях, которые требуют доступа к исходным данным.

Вместо сквозного шифрования всей базы данных может быть целесообразнее использовать частичное шифрование, где шифруются только конкретные конфиденциальные данные или поля, сохраняя остальные данные в незашифрованном виде. Это позволяет более гибко балансировать безопасность и производительность системы. Именно так мы и поступим. Будем шифровать только те данные, которые могут определенно точно идентифицировать личность человека.

Воспользуемся модулем `pgcrypto` для симметричного шифрования выборочных данных. Но для начала убедимся, что модуль установлен. На рисунке 15 можно видеть, что у нас все готово к работе.

Query History	
1	SELECT extname FROM pg_extension;
Data Output Notifications Messages	
	extname name
1	plpgsql
2	adminpack
3	pgcrypto

Рисунок 15 – Проверка наличия модуля pgcrypto

Можно шифровать и расшифровывать столбцы в таблицах в созданной БД:

- для шифрования используется следующий шаблон: «UPDATE mytable SET mycolumn = pgp_sym_encrypt(mycolumn, 'mysecretkey');»;
- для расшифрования используется следующий шаблон: «SELECT pgp_sym_decrypt(mycolumn, 'mysecretkey') AS decrypted_data FROM mytable;».

На рисунке 16 приведены данные в чистом виде без шифрования.

Query History	
1	SELECT last_name, first_name, middle_name, phone_number, passport, date_of_birth
2	FROM user_profile
3	ORDER BY user_profile.id ASC;
Data Output Notifications Messages	
	last_name character varying (50) first_name character varying (50) middle_name character varying (50) phone_number character varying (50) passport character varying (50) date_of_birth date
1	Иванов Иван Иванович 88005553535 5353 535353 1978-04-04

Рисунок 16 – Изначальные данные таблицы user_profile

На рисунке 17 представлен пример того, как шифруются некоторые столбцы.

```

Query   Query History
1 UPDATE user_profile SET date_of_birth = pgp_sym_encrypt(date_of_birth, 'encryption_key');
2 UPDATE user_profile SET phone_number = pgp_sym_encrypt(phone_number, 'encryption_key');
3 UPDATE user_profile SET passport = pgp_sym_encrypt(passport, 'encryption_key');

Data Output  Notifications  Messages

```

UPDATE 1

Query returned successfully in 53 msec.

Рисунок 17 – Шифрование некоторых данных таблицы user_profile

Теперь убедимся, что наши данные и правда были зашифрованы. Это можно увидеть на рисунке 18.

```

y   Query History
SELECT last_name, first_name, middle_name, phone_number, passport, date_of_birth
FROM user_profile
ORDER BY id ASC;

Output  Notifications  Messages

```

last_name	first_name	middle_name	phone_number
character varying (50)	character varying (50)	character varying (50)	text
Иванов	Иван	Иванович	\xc30d04070302a2f3c446188c7e3976d2

Рисунок 18 – Проверка шифрования некоторых данных таблицы user_profile

На рисунке 18 видно, что столбец с паспортными данными успешно зашифрован и можно не переживать за конфиденциальность хранения персональных данных. Но что теперь делать, нужно же не только хранить, но и получать данные из таблицы. Для этого при выполнении запроса вызывается функция расшифрования с тем самым ключом, который использовался при шифровании. На рисунке 19 продемонстрирован запрос на получения данных в зашифрованного столбца с процессом расшифрования по ходу выполнения.

Query		Query History	
1		SELECT pgp_sym_decrypt(passport, 'encryption_key') AS passport FROM user_profile;	
Data Output		Notifications	
	passport character varying (50)		
1	5353 535353		

Рисунок 19 – Получение данных зашифрованного столбца с расшифрованием в процессе выполнения запроса

В конечном итоге, после расшифрования всех данных, мы получим идентичные данные, что были до процесса шифрования. На рисунке 20 можно увидеть, что целостность данных не была нарушена после процессов шифрования и дешифрования данных.

Query		Query History	
1		SELECT last_name, first_name, middle_name, phone_number, passport, date_of_birth 2 FROM user_profile 3 ORDER BY user_profile.id ASC;	
Data Output		Notifications	
	last_name character varying (50)	first_name character varying (50)	middle_name character varying (50)
	phone_number character varying (50)	passport character varying (50)	date_of_birth date
1	Иванов	Иван	Иванович
	88005553535	5353 535353	1978-04-04

Рисунок 20 – Подтверждение целостности данных

3.5 Порты

Порт – это некоторое идентифицирующее число от 1 до 65535, позволяющее протоколам взаимодействовать друг с другом на транспортном уровне модели OSI.

На самом деле, у PostgreSQL сервера по умолчанию установлен только один порт, и это TCP порт.

PostgreSQL использует TCP/IP для обмена данными между клиентами и сервером. По умолчанию, порт TCP для PostgreSQL установлен на 5432. TCP 5432: database engine, используется для подключения к СУБД. Этот порт можно изменить при настройке сервера [17].

Несмотря на то, что UDP не используется для коммуникации между клиентами и сервером PostgreSQL, есть несколько расширений, таких как pgpool-II, которые могут использовать UDP для взаимодействия между

серверами PostgreSQL.

Таким образом, можно сказать, что PostgreSQL сервер по умолчанию использует только один порт TCP для обмена данными между клиентами и сервером.

Хорошей практикой является смена портов на другие, чтобы при случае атаки порты пришлось перебирать методом грубого подбора, а не использовать установленные по умолчанию. По данным IANA, порты 834-846 являются свободными, поэтому мы их можем использовать.

Для изменения порта, используемого PostgreSQL, можно внести изменения в конфигурационный файл `postgresql.conf`, который находится в каталоге данных PostgreSQL.

В файле `postgresql.conf` необходимо изменить параметр `port` на желаемое значение порта. Например, если вы хотите использовать порт 840 вместо стандартного порта 5432, необходимо изменить строку: `port = 5432` на `port = 840`. После этого нужно перезапустить PostgreSQL, чтобы изменения вступили в силу.

Также можно использовать параметр командной строки `-p` при запуске сервера PostgreSQL для указания порта. Например, чтобы запустить PostgreSQL на порту 840, необходимо использовать команду: `postgres -p 840`, но в этом случае необходимо убедиться, что порт 840 свободен и не используется другим приложением на сервере.

Помимо смены порта, слушающий входящие соединения, можно отключить PostgreSQL Server. PostgreSQL Server – это одна из служб PostgreSQL, отвечающая за прослушивание запросов. Если эту службу отменить, то для установления соединения придется явно указывать номер порта, что обеспечивает дополнительную безопасность. Для остановки этой службы требуется просто ввести `sudo systemctl stop postgresql` в командной строке.

3.6 Брандмауэр

Брандмауэр – это программный межсетевой экран, имеющийся в ОС Windows. В свою очередь межсетевой экран – это элемент локальной сети, осуществляющий мониторинг активности в данной сети, а именно фильтрацию сетевого трафика по заранее описанным правилам. PostgreSQL Server функционирует на ОС Windows, поэтому рекомендуется использовать

Брандмауэр вместе с программно-аппаратным межсетевым экраном. Такое сочетание позволяет повысить уровень защищенности за счёт использования дополнительных механизмов защиты информации. Также Брандмауэр обеспечивает не только защищенность от НСД, но и от поступающего вредоносного трафика в принципе.

Как было описано выше, межсетевой экран анализирует сетевой трафик и отклоняет его, если он считается подозрительным. Выше мы поменяли стандартный TCP/UDP порт 5432 на 840, теперь нужно объявить правило в межсетевом экране, что публичная сеть не может ссылаться на этот порт. Для этого заходим в Монитор Брандмауэра Защитника Windows в режиме повышенной безопасности и создадим новое правило. Далее определяем, что правило создается для порта, затем вписываем наш нужный порт, после этого разрешаем подключение, после этого исключаем публичный доступ, и в конце сохраняем наше правило.

Выводы по разделу

В этом разделе разбирались аспекты, связанные с безопасностью. Для защиты от НСД внутренними средствами PostgreSQL была реализована авторизация пользователей, а также им были назначены необходимые минимальные права для работы с БД, что позволило осуществить легитимный доступ к информации. Для повышения конфиденциальности информации использовалось шифрование информации на уровне столбцов при помощи библиотеки pgcrypto. Кроме того, при помощи шифрования файлы БД были защищены от кражи на физических носителях. А изменение портов и использование Брандмауэра позволило защититься на сетевом уровне. В совокупности была выполнена задача определения защитных мер БД.

4 ТЕСТИРОВАНИЕ И ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ

4.1 Методы и инструменты тестирования

Перед заключительным этапом проводят тестирование базы данных и ее оптимизацию. На этом этапе производятся проверки корректной и эффективной работы базы данных. Кроме того, выполняются тесты производительности, определяются проблемы с ней и соответственно решение этих проблем путем оптимизации запросов, индексов и настройки СУБД PostgreSQL.

Для тестирования базы данных можно использовать различные методы и инструменты, такие как:

Существуют различные методы и инструменты для проверки базы данных, включая:

- модульное тестирование: это проверка отдельных компонентов базы данных, таких как хранимые процедуры, триггеры, функции и другие;
- интеграционное тестирование: это проверка совместной работы базы данных с приложением, которое использует эту базу данных;
- нагрузочное тестирование: это проверка производительности базы данных в условиях высокой нагрузки;
- тестирование безопасности: это проверка уязвимостей базы данных и соответствия ее требованиям безопасности;
- ручное тестирование: это проверка базы данных на соответствие требованиям и наличие ошибок, которую выполняет тестировщик вручную.

4.2 Нагрузочное тестирование и оптимизация производительности

Нагрузочное тестирование БД в PostgreSQL - это процесс измерения производительности и стабильности базы данных при большой нагрузке. Это необходимо для определения максимального количества пользователей и запросов, которые могут быть обработаны базой данных, без ухудшения ее производительности.

Для проведения нагрузочного тестирования БД в PostgreSQL,

необходимо выполнить следующие шаги:

- определить сценарии использования БД: определить типы запросов, которые будут отправлены к БД, и количество пользователей, которые будут использовать приложение одновременно;
- подготовить тестовые данные: создать тестовые данные, которые будут использоваться при выполнении запросов в тестовом окружении. Важно создать реалистичные тестовые данные, чтобы результаты нагрузочного тестирования были максимально близки к реальной нагрузке;
- настроить окружение для нагрузочного тестирования: убедиться, что база данных и приложение настроены правильно, чтобы обеспечить максимальную производительность и стабильность;
- запустить нагрузочное тестирование: выполнить тестовые сценарии использования БД в тестовом окружении и измерить производительность базы данных, используя метрики, такие как скорость ответа, время выполнения запросов, количество ошибок и использование ресурсов;
- анализ результатов: проанализировать результаты нагрузочного тестирования, чтобы определить максимальную нагрузку, которую может выдержать база данных, и определить узкие места, которые могут быть оптимизированы.

При тестировании в PostgreSQL можно использовать специальные инструменты, такие как pgbench и Apache JMeter, чтобы автоматизировать процесс выполнения тестовых сценариев и сбора метрик производительности. Важно понимать, что результаты нагрузочного тестирования могут изменяться в зависимости от настроек и конфигурации сервера и базы данных, поэтому рекомендуется проводить тестирование на реальном оборудовании, которое будет использоваться в итоговой среде.

- pgbench: это инструмент, входящий в стандартный комплект поставки PostgreSQL. Он предназначен для проведения нагрузочного тестирования на уровне SQL-запросов [18];
- pgBadger: это инструмент для анализа логов PostgreSQL. Он позволяет анализировать логи запросов и генерировать отчеты о производительности базы данных;
- HammerDB: это инструмент для тестирования производительности

баз данных. Он поддерживает PostgreSQL и позволяет тестировать производительность базы данных на различных уровнях нагрузки;

- Sysbench: это инструмент, который может быть использован для тестирования производительности базы данных PostgreSQL. Он позволяет тестировать производительность базы данных на уровне SQL-запросов и многопоточности.

Важно помнить, что результаты нагрузочного тестирования БД PostgreSQL могут зависеть от конкретной конфигурации сервера и структуры данных, а также от условий использования базы данных в реальном мире. Поэтому результаты тестирования должны быть интерпретированы с осторожностью и использованы для оптимизации конкретной базы данных.

Оптимизация производительности базы данных PostgreSQL может включать в себя ряд мероприятий, чтобы ускорить выполнение SQL запросов и обеспечить более эффективную работу БД. Вот основные подходы, которые можно эффективно использовать для оптимизации БД PostgreSQL [19; 20]:

- создание правильных индексов: индексы ускоряют выполнение запросов, позволяя PostgreSQL быстро находить нужные данные. Убедитесь, что у вас есть индексы на часто используемые столбцы в запросах, а также на столбцы, используемые в условиях WHERE и JOIN и фильтрации данных, может значительно ускорить выполнение запросов;
- оптимизация запросов: оптимизация запросов, такая как использование подзапросов, объединения и группировки данных, может улучшить производительность запросов и уменьшить количество запросов, необходимых для выполнения операции;
- разделение данных на отдельные таблицы: если у нас есть большие таблицы, то не следует хранить большие объемы данных в одной таблице. Разделение этих данных на отдельные таблицы или партицирование может помочь ускорить выполнение запросов. Это позволяет более эффективно управлять объемом данных, используемых при выполнении запросов;
- оптимизация схемы базы данных: иногда изменение структуры базы данных может привести к улучшению производительности. Например, использование более эффективных типов данных, уменьшение количества NULL значений или устранение

- избыточных таблиц и связей;
- использование кэширования: если важна скорость выполнения, то можно использовать кэширование для часто запрашиваемых данных. Использование кэширования запросов может значительно улучшить производительность, позволяя избежать выполнения сложных запросов, возвращая результаты из кэша. Это особенно полезно для запросов, которые выполняются часто и имеют статические результаты;
- оптимизация запросов на запись: если есть интенсивная нагрузка на запись данных, то различные техники, такие как пакетная вставка (`batch insert`), использование транзакций и пакетных обновлений, могут помочь ускорить процесс записи данных;
- предварительная компиляция запросов: в PostgreSQL есть возможность предварительной компиляции запросов (`prepared statements`), что позволяет повторно использовать выполненные запросы и сократить накладные расходы на компиляцию. Предварительную компиляцию стоит использовать для запросов, которые выполняются многократно с различными параметрами;
- настройка конфигурации PostgreSQL: немаловажное значение имеет изучение и настройка параметров конфигурации PostgreSQL в соответствии с требованиями конкретной системы. Некоторые параметры, которые можно настроить, включают размер буферов, параллелизм выполнения запросов и максимальное количество одновременных соединений;
- обновление до последней версии PostgreSQL: немаловажное значение имеет актуальность поддерживаемого ПО, поэтому рекомендуется использовать последнюю стабильную версию PostgreSQL. Каждое новое обновление может содержать оптимизации и улучшения производительности, которые могут существенно повысить скорость выполнения запросов;
- настройка памяти и дискового пространства: правильная настройка памяти и дискового пространства может существенно повлиять на производительность PostgreSQL. Следует выделять достаточное количество памяти для работы с базой данных. Кроме того, система должна иметь достаточное дисковое пространство для хранения

- данных и временных файлов;
- использование материализованных представлений: материализованные представления - это предварительно вычисленные результаты запросов, сохраняемые в виде таблиц. Они могут быть особенно полезны, если имеются сложные запросы с большими объемами данных, которые выполняются часто. Материализованные представления позволяют значительно сократить время выполнения запросов;
- настройка параллелизма: PostgreSQL поддерживает параллельное выполнение запросов, что может значительно ускорить обработку больших объемов данных. Можно настроить параметры параллелизма в зависимости от характеристик системы и требований к производительности;
- горизонтальное масштабирование: если даже после оптимизации базы данных путем проведения вышеперечисленных манипуляций все еще возникают проблемы с производительностью , можно рассмотреть вариант горизонтального масштабирования. Распределение данных на несколько серверов может помочь справиться с высокой нагрузкой и улучшить производительность;
- оптимизация настройки сервера: настройка параметров сервера PostgreSQL, таких как размер буфера и количество параллельных запросов, может улучшить производительность базы данных.

Важно проводить тестирование на реалистичных тестовых данных и сценариях использования, чтобы получить наиболее точные результаты. Также при проведении нагрузочного тестирования БД PostgreSQL необходимо учитывать следующие факторы:

- объем данных: чем больше объем данных в базе данных, тем больше ресурсов требуется для обработки запросов. Поэтому важно учитывать объем данных при выборе конфигурации сервера и оптимизации производительности;
- конфигурация сервера: настройка сервера PostgreSQL имеет большое значение для производительности базы данных. Например, увеличение размера буфера может ускорить выполнение запросов, но при этом может потребовать больше памяти;
- структура данных: структура таблиц и связей между ними может

влиять на производительность базы данных. Например, использование ненормализованных таблиц может привести к медленной работе базы данных;

- распределение запросов: распределение запросов между разными серверами или узлами может увеличить производительность и снизить нагрузку на базу данных,

С учётом вышеперечисленных возможностей для оптимизации и тестирования, было проведено нагрузочное тестирование полностью готовой, оптимизированной и заполненной реальными данными БД для полноценного функционирования web-приложения opmp.ru с помощью модуля PostgreSQL - pgbench.

Для проведения нагрузочного тестирования требуется:

- инициализировать базу данных с помощью pgbench. Это создаст необходимые таблицы и заполнит их тестовыми данными:
«pgbench -i -s 100 database_name».

Флаг -s 100 указывает на размер масштабируемости данных, где 100 означает, что будет создано в 100 раз больше строк, чем количество таблиц. Можно изменить это значение в соответствии с потребностями в тестировании;

- запускаем нагрузочное тестирование с помощью pgbench. Ниже приведен пример команды, которая запускает тест с 10 параллельными клиентами и общим числом транзакций 1000:
«pgbench -c 10 -j 10 -t 1000 database_name».

Флаг -c 10 указывает на количество параллельных клиентов, -j 10 указывает на количество параллельных потоков (обычно выбирают такое же значение, как и -c), а -t 1000 указывает на общее количество транзакций для выполнения;

- далее анализируем результаты тестирования, которые будут выведены в консоль. Они включают среднее время выполнения транзакции, количество транзакций в секунду и другие метрики производительности;
- затем будем просто изменять параметры команды pgbench, такие как количество клиентов, количество потоков и общее количество транзакций, чтобы провести более интенсивное или длительное тестирование.

С подробными результатами нагрузочного тестирования можно ознакомиться в приложении Б. По полученным результатам можно увидеть и смело сделать вывод, что разработанная база данных в полной мере выдерживает нагрузки при больших объемах хранимых и обрабатываемых данных. Кроме того, при одновременном подключении порядка 50-80 пользователей, система выдержала нагрузку и задержка на подключение пользователя к сессии составляет всего около 0.5 секунд, что никак отрицательно не отражается на работоспособности и корректном функционировании web-приложения.

Тестирование БД PostgreSQL является важным этапом при разработке и оптимизации базы данных. Оно позволяет выявить узкие места в работе базы данных и произвести необходимую оптимизацию, чтобы обеспечить максимальную производительность и стабильность приложения.

Стоит отметить, что после внедрения базы данных необходимо проводить анализ ее использования и результата. Это позволит выявить потенциальные улучшения и оптимизации процессов работы с медицинской информацией. На основе анализа можно внести изменения в структуру базы данных, внедрить новые функциональные возможности или улучшить существующие.

4.3 Выполнение запросов

После того, как мы разобрались с видами тестирования, оптимизацией памяти и скорости выполнения запросов, самое время продемонстрировать полученные результаты.

На рисунке 21 представлен SQL запрос к таблицам медикаментов, представленных на рисунке 7.

```
SELECT DISTINCT
medicines.name AS "Название препарата Им.п.",
medicines.name_genitive AS "Название препарата Р.п.",
medicines.unit AS "Мера медикамента",
diag.name AS "Название диагноза",
child_dosages.dosage AS "Детская дозировка",
child_dosages.unit AS "Мера детской дозировки",
medicines.contraindications AS "Противопоказания препарата"
FROM medicines
LEFT JOIN child_dosages ON medicines.id = child_dosages.medicines_id
JOIN diag ON child_dosages.diag_id = diag.id
ORDER BY medicines.name ASC, diag.name ASC;
```

Рисунок 21 – SQL запрос получения всех медикаментов

Запрос есть, теперь нужно посмотреть корректность выводимых данных.
На рисунке 22 представлен вывод SQL запроса с рисунка 21.

Название препарата Им.п.	Название препарата Р.п.	Мера медикамента
S. Aminophyllinum 24 mg/ml	S. Aminophyllini 24 mg/ml	mg
S. Amiodaronum 50 mg/ml	S. Amiodaroni 50 mg/ml	mg
S. Atropinum 1 mg/ml	S. Atropini 1 mg/ml	mg
S. Chloropyraminum 20 mg/ml	S. Chloropyramini 20 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Dexametasonum 4 mg/ml	S. Dexametasoni 4 mg/ml	mg
S. Diphenhydraminum 10 mg/ml	S. Diphenhydramini 10 mg/ml	mg
S. Dorzolamidum 20 mg/ml	S. Dorzolamidi 20 mg/ml	blob
S. Drotaverinum 20 mg/ml	S. Drotaverini 20 mg/ml	mg
S. Epinephrinum 1 mg/ml	S. Epinephrini 1 mg/ml	mg
Название диагноза	Детская дозировка	Мера детской дозировки
Общая дозировка	4,0	weight
Общая дозировка	5,0	weight
Общая дозировка	0,01	weight
Общая дозировка	0,1	age
Инфекционно-токсичес...	0,6	weight
Менингит, вызванный ...	0,6	weight
Менингококкемия	0,6	weight
Острый бронхит, бронх...	0,2	weight
Острый обструктивный...	0,6	weight
Острый обструктивный...	0,2	weight
Острый обструктивный...	0,6	weight
Отек гортани (аллерги...	0,2	weight
Отек гортани (аллерги...	0,6	weight
Отравлением калием п...	0,6	weight
Отравление препарата...	0,6	weight
Термические и химиче...	0,6	weight
Общая дозировка	0,1	weight
Общая дозировка	1,0	blob
Общая дозировка	0,1	age
Остановка сердца	0,1	weight
Противопоказания препарата		
Гиперчувствительность; Острый инфа...		
Синусовая брадикардия; СССУ; Синоа...		
Повышенная чувствительность; Закр...		
Гиперчувствительность; Закрытоугол...		
Гиперчувствительность; Системные г...		
Гиперчувствительность; Кормление г...		
Гиперчувствительность		
Гиперчувствительность; Гиперчувст...		
Повышенная чувствительность; Гипе...		

Рисунок 22 – Результаты запроса к таблицам медикаментов

Все результаты моей деятельности - проектирования, выполнения запросов и их оптимизация, впоследствии передаются на бэкенд. На рисунке 23 представлен JSON формат по отображению всех медикаментов со взрослыми и детскими дозировками из данных, полученных при обращении к БД.

```

▼ S. Aminophyllinum 24 mg/ml:
    genitive:                      "S. Aminophyllini 24 mg/ml"
    unit:                           "mg"
    ▼ diagnoses:
        ▼ Общая дозировка:
            adult dosage:           240
            child dosage:           4
    ▼ contraindications:
        0:                            "Гиперчувствительность"
        1:                            "Острый инфаркт миокарда"
        2:                            "Первая половина беременности"
        child dosage unit:          "weight"
► S. Amiodaronum 50 mg/ml:
▼ S. Atropinum 1 mg/ml:
    genitive:                      "S. Atropini 1 mg/ml"
    unit:                           "mg"
    ▼ diagnoses:
        ▼ Общая дозировка:
            adult dosage:           0.5
            child dosage:            0.01
    ▼ contraindications:
        0:                            "Повышенная чувствительность"
        ▶ 1:                           "Закрытоугольная глаукома...едрасположенность к ней"
        ▶ 2:                           "Тахикардии, тяжелая хром... ИБС, митральный стеноз"
        ▶ 3:                           "Рефлюкс-эзофагит, ГПОД, стеноз привратника"
        ▶ 4:                           "Печеночная и/или почечная недостаточность"
        ▶ 5:                           "Атония кишечника"
        ▶ 6:                           "Миастения"
        ▶ 7:                           "Задержка мочи или предрасположенность к ней"
        ▶ 8:                           "Заболевания сопровождающ...ией мочевыводящих путей"
        ▶ 9:                           "Обструктивные заболевани...фический язвенный колит"
        10:                          "Ксеростомия"
        11:                          "Миастения"
        12:                          "Болезнь Дауна"
        13:                          "ДЦП"
        14:                          "Беременность, осложненная гестозом"
        15:                          "ГПОД"
        16:                          "Период грудного вскармливания"
        child dosage unit:          "weight"

```

Рисунок 23 – JSON формат данных, полученных при обращении бэкенда к таблицам медикаментов

На рисунке 24 представлен SQL запрос к таблицам заболеваний, представленных на рисунке 6.

```

SELECT DISTINCT diseases.id, disease_symptoms.id, form_symptoms.id,
diseases.name AS "Заболевание",
diseases.description AS "Описание заболевания",
diseases.period AS "Период заболевания",
disease_symptoms.name AS "Симптомы заболевания",
form_disease.name AS "Формы заболевания",
form_disease.description AS "Описание формы заболевания",
form_symptoms.name AS "Симптомы формы заболевания"
FROM diseases
JOIN disease_tags ON diseases.tag_id = disease_tags.id
LEFT JOIN diseases_disease_symptoms ON diseases.id = diseases_disease_symptoms.diseases_id
LEFT JOIN disease_symptoms ON diseases_disease_symptoms.symptom_diseases_id = disease_symptoms.id
LEFT JOIN diseases_form_disease ON diseases.id = diseases_form_disease.diseases_id
LEFT JOIN form_disease ON diseases_form_disease.form_id = form_disease.id
LEFT JOIN form_disease_form_symptoms ON form_disease.id = form_disease_form_symptoms.form_id
LEFT JOIN form_symptoms ON form_disease_form_symptoms.symptom_form_id = form_symptoms.id
WHERE disease_tags.name_tag ILIKE %
ORDER BY diseases.id ASC, disease_symptoms.id ASC, form_symptoms.id ASC;

```

Рисунок 24 – SQL запрос по части названия заболевания

На рисунке 25 представлен вывод SQL запроса с рисунка 24.

15 12\16	Чума	Чума — это особо опасная ... Инкубационный период... Интоксикация: повышение ...
15 12\17	Чума	Чума — это особо опасная ... Инкубационный период... Интоксикация: повышение ...
15 12\18	Чума	Чума — это особо опасная ... Инкубационный период... Интоксикация: повышение ...
15 12\19	Чума	Чума — это особо опасная ... Инкубационный период... Интоксикация: повышение ...
16 12\20	Сибирская язва	Сибирская язва — это особ... Инкубационный период... Интоксикация при всех фор...
16 12\21	Сибирская язва	Сибирская язва — это особ... Инкубационный период... Интоксикация при всех фор...
16 12\22	Сибирская язва	Сибирская язва — это особ... Инкубационный период... Интоксикация при всех фор...
16 12\23	Сибирская язва	Сибирская язва — это особ... Инкубационный период... Интоксикация при всех фор...
16 12\24	Сибирская язва	Сибирская язва — это особ... Инкубационный период... Интоксикация при всех фор...
16 12\25	Сибирская язва	Сибирская язва — это особ... Инкубационный период... Интоксикация при всех фор...
16 12\20	Сибирская язва	Сибирская язва — это особ... Инкубационный период... ЧСС >100 уд./мин.
16 12\21	Сибирская язва	Сибирская язва — это особ... Инкубационный период... ЧСС >100 уд./мин.
16 12\22	Сибирская язва	Сибирская язва — это особ... Инкубационный период... ЧСС >100 уд./мин.
16 12\23	Сибирская язва	Сибирская язва — это особ... Инкубационный период... ЧСС >100 уд./мин.
16 12\24	Сибирская язва	Сибирская язва — это особ... Инкубационный период... ЧСС >100 уд./мин.
16 12\25	Сибирская язва	Сибирская язва — это особ... Инкубационный период... ЧСС >100 уд./мин.
16 12\20	Сибирская язва	Сибирская язва — это особ... Инкубационный период... АД <100/60 мм рт.ст.
16 12\21	Сибирская язва	Сибирская язва — это особ... Инкубационный период... АД <100/60 мм рт.ст.
16 12\22	Сибирская язва	Сибирская язва — это особ... Инкубационный период... АД <100/60 мм рт.ст.
16 12\23	Сибирская язва	Сибирская язва — это особ... Инкубационный период... АД <100/60 мм рт.ст.
16 12\24	Сибирская язва	Сибирская язва — это особ... Инкубационный период... АД <100/60 мм рт.ст.
	Легочная	Легочно-сердечная недостаточность.
	Кишечная	Боли в эпигастрии, животе, сильные; ...
	Септическая	Сыпь геморрагическая на коже, слиз...
	Септическая	Рвота с примесью крови.
	Кожная	Медно-красное пятно, папула, жжени...
	Кожная	Лимфаденит.
	Карбункулезная	Отек кожи, некроз, карбункулы, гемо...
	Карбункулезная	Отечная, гиперемированная, но безбо...
	Септическая	Сыпь геморрагическая на коже, слиз...
	Желудочно-кишечная	Рвота и фекалии с примесью крови.
	Кожная	Медно-красное пятно, папула, жжени...
	Кожная	Лимфаденит.
	Карбункулезная	Отек кожи, некроз, карбункулы, гемо...
	Карбункулезная	Отечная, гиперемированная, но безбо...
	Септическая	Сыпь геморрагическая на коже, слиз...
	Желудочно-кишечная	Рвота и фекалии с примесью крови.
	Кожная	Медно-красное пятно, папула, жжени...
	Кожная	Лимфаденит.
	Карбункулезная	Отек кожи, некроз, карбункулы, гемо...
	Карбункулезная	Отечная, гиперемированная, но безбо...
	Септическая	Сыпь геморрагическая на коже, слиз...

Рисунок 25 – Результаты запроса к таблицам заболеваний

На рисунке 26 представлен JSON формат по отображению всех заболеваний из данных, полученных при обращении к БД.

▼ Мalaria:	
tag:	"Инфекционные заболевания"
description:	"Мalaria – острое инфекционно-вирусное заболевание с цианотичным оттенком."
period:	null
▼ symptoms:	
0:	"Сильная головная и мышечная боль."
1:	"Кожа бледная, холодная, с цианотичным оттенком."
2:	"Жажда, иногда рвота; бред."
3:	"Тахикардия."
4:	"Лицо гиперемировано, кожа сухая, горячая."
5:	"Через несколько часов жаофузным потоотделением."
6:	"Температура тела критично-субнормальных значений."
7:	"Самочувствие улучшается, но остается слабость."
8:	"Потрясающий озноб от 10-ратуры тела до 39-40°C."
▼ forms:	
0:	null
▼ form descriptions:	
0:	null
▼ form symptoms:	
0:	null
▼ Ботулизм:	
tag:	"Инфекционные заболевания"
description:	"Ботулизм – наиболее тяжелых и дыхательных мышц."
period:	"Инкубационный период: от 3-12-24 ч до 2-5-14 дней."
▼ symptoms:	
0:	"Жажда; сухость слизистых-кожных покровов; озноб."
1:	"Рвота; понос более 3 раз."
2:	"Акроцианоз."
3:	"Ввалившиеся глаза."
4:	"Впалые щеки."
5:	"Обесцвеченный (белый) кал."
6:	"Выраженная общая слабость; пространия."
7:	"Темная моча."
8:	"Снижение АД; частый пульс."
9:	"ИТШ."
10:	"Олигурия, анурия; судороги."
11:	"Афония – хриплый голос."
12:	"Снижение тургора кожи."
▼ forms:	
0:	null
▼ form descriptions:	
0:	null
▼ form symptoms:	
0:	null
► Скарлатина:	{...}
► Гепатит вирусный:	{...}
► Коронавирусная инфекция:	{...}
► Корь:	{...}
► Дифтерия:	{...}
► Менингококковая инфекция, менингит, менингококциемия:	{...}

Рисунок 26 – JSON формат данных, полученных при обращении бэкенда к таблицам заболеваний

На рисунках выше были приведены примеры запросов и полученные данные основных таблиц: заболевания, диагнозы и медикаменты. Но кроме основных, еще имеются вспомогательные таблицы дифференциальной диагностики, которые носят информативно-вспомогательный характер.

На рисунке 27 представлены SQL запросы к таблицам дифференциальной диагностики для корректного взаимодействия с бэкендом.

```
COMA_SCALE_EYE_REACTIONS_SQL = \
...
SELECT DISTINCT sign AS "Признак", points AS "Баллы"
FROM coma_scale_eye_reactions
ORDER BY coma_scale_eye_reactions.points DESC;
...
COMA_SCALE_MOTORREACTIONS_SQL = \
...
SELECT DISTINCT sign AS "Признак", points AS "Баллы"
FROM coma_scale_motor_reactions
ORDER BY coma_scale_motor_reactions.points DESC;
...
COMA_SCALE_STEM_REFLEXES_SQL = \
...
SELECT DISTINCT sign AS "Признак", points AS "Баллы"
FROM coma_scale_stem_reflexes
ORDER BY coma_scale_stem_reflexes.points DESC;
...
COMA_SCALE_BREATHING_PATTERN_SQL = \
...
SELECT DISTINCT sign AS "Признак", points AS "Баллы"
FROM coma_scale_breathing_pattern
ORDER BY coma_scale_breathing_pattern.points DESC;
...
COMA_SCALE_INTERPRETATION_SQL = \
...
SELECT DISTINCT point AS "Балл", sign AS "Признак"
FROM coma_scale_interpretation
ORDER BY coma_scale_interpretation.point DESC;
...
```

Рисунок 27 – SQL запросы к таблицам дифференциальной диагностики

На рисунке 28 представлен JSON формат таблиц дифференциальной диагностики из данных, полученных при обращении бэкенда к БД.

▶ Шкала Глазго (Glasgow Coma Scale):	{...}
▼ Шкала комы FOUR:	
▼ data:	
▼ Глазные реакции (E):	
▼ 0:	
Признак:	"Глаза открыты слежение и мигание по команде"
Баллы:	4
▼ 1:	
Признак:	"Глаза открыты, но нет слежения"
Баллы:	3
▼ 2:	
Признак:	"Глаза закрыты, открываютъ звук, но слежения нет"
Баллы:	2
▼ 3:	
Признак:	"Глаза закрыты, открываютъ боль, но слежения нет"
Баллы:	1
▼ 4:	
Признак:	"Глаза остаются закрытыми в ответ на боль"
Баллы:	0
▶ Двигательные реакции (M):	[...]
▶ Стволовые рефлексы (B):	[...]
▶ Дыхательный паттерн (R):	[...]
▶ Интерпретация результата:	[...]
part_name:	""
note:	""
type_table:	4
type_result:	7
▶ Шкала моторного дефицита LAMS (Los Angeles Motor Scale):	{...}

Рисунок 28 – JSON формат данных, полученных при обращении к таблицам заболеваний

На рисунке 29 представлен JSON формат всех таблиц дифференциальной диагностики.

▶ Акушерство:	{...}
▶ Нормы ЧД, ЧСС, АД у детей (в покое):	{...}
▶ Параметры проведения базовой СЛР:	{...}
▶ Промывание желудка у детей:	{...}
▶ Размеры эндотрахеальных трубок у детей:	{...}
▶ Соответствие размеров ларингеальных трубок параметрам пациента:	{...}
▶ ВАШ, НОШ. Шкалы оценки интенсивности боли:	{...}
▶ Острая дыхательная недостаточность (Кассиль В.Л. 2004 г.):	{...}
▶ Оценка мышечной силы по баллам:	{...}
▶ Шкала возбуждения-седации Ричмонда (шкала RASS):	{...}
▶ Определение площади ожогов у детей (по Lund и Browder):	{...}
▶ Шкала оценки вероятности ТЭЛА (Revised Geneva Score):	{...}
▶ Критерии оценки новорождённого по шкале Апгар:	{...}
▶ Протокол оценки тяжести состояния пациента (NEWS):	{...}
▶ ХСН ШОКС (в модификации Мареева В.Ю.):	{...}
▶ Шкала Глазго (Glasgow Coma Scale):	{...}
▶ Шкала комы FOUR:	{...}
▶ Шкала моторного дефицита LAMS (Los Angeles Motor Scale):	{...}

Рисунок 29 – Таблицы дифференциальной диагностики

Выводы по разделу

Тестирование базы данных является критически важным этапом в разработке и поддержке приложений, использующих базы данных. Тестирование базы данных позволяет убедиться в корректности и надежности ее работы, а также обеспечить защиту от ошибок и нарушений безопасности.

Использование различных методов и инструментов позволяет выявить проблемы в базе данных еще на ранних стадиях и устраниить их до того, как они приведут к серьезным проблемам в работе приложения. Кроме того, тестирование базы данных помогает повысить уровень доверия пользователей к приложению и защитить данные от нарушений и утечек. В целом, проведение тестирования базы данных является критически важным для обеспечения надежности, безопасности и производительности приложений.

ЗАКЛЮЧЕНИЕ

В выпускной квалификационной работе бакалавра были рассмотрены особенности внедрения цифровых технологий в работу отделения неотложной медицинской помощи, а также разработана система электронных медицинских карт на базе PostgreSQL, которая позволяет улучшить качество и эффективность работы скорой медицинской помощи.

Был проведен анализ системы медицинских карт и баз данных на языке PostgreSQL, а также их применение в работе скорой медицинской помощи. Также были рассмотрены вопросы обеспечения безопасности хранения медицинских данных и конфиденциальности пациентов при использовании электронных медицинских карт.

Помимо БД, имеющей нормализованную структуру, была проведена работа по её защите. Защита строилась как внутренними механизмами и функционалом PostgreSQL Server, так и посредством использования встроенной опциональности ОС.

В работе использовался PostgreSQL для хранения всех возможных данных пациентов. Это позволило создать систему, которая может автоматизировать процесс сбора и обработки данных, а также обеспечивать быстрый и безопасный доступ к истории болезней и личным данным пациентов.

БД спроектирована и защищена, поэтому её можно вводить в эксплуатацию. По желанию в неё можно добавить дополнительный функционал, состоящий из: представлений, индексов, функций и процедур. Все эти объекты БД может настроить администратор или лицо, обслуживающее её, по желанию сотрудников, работающих с ней. Основной функционал и защита были выполнены исходя из целей, а именно:

- изучения особенностей предметной области;
- проектирования логических и физических моделей данных;
- разработки структуры базы данных;
- нормализация данных;
- определения и настройки защитных мер;
- обеспечения конфиденциальности хранения персональных данных;
- проведение тестирования, оптимизации запросов и структуры базы данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Worsley J., Drake J. Practical PostgreSQL. — New York, USA : O'Reilly Media, Inc., 2002. — 640 с.
2. Кириллов В. В., Громов Г. Ю. Введение в реляционные базы данных. — Санкт-Петербург : БХВ-Петербург, 2009. — 454 с.
3. НОУ_ИНТУИТ. Основные понятия баз данных. — 2021. — URL: <https://intuit.ru/studies/courses/4426/681/lecture/14017> (дата обращения 15.02.2023).
4. Obe R., Hsu L. PostgreSQL: Up and Running. — New York, USA : O'Reilly Media, Inc., 2014. — 232 с.
5. Ahmed I., Smith G., Pirozzi E. PostgreSQL High Performance. — Birmingham : Packt Publishing Ltd, 2018. — 508 с.
6. Голицына О. Л., Партика Т. Л., Попов И. И. Основы проектирования баз данных. — Москва : Издательство ФОРУМ, 2021. — 416 с.
7. StudFile. Концепция баз данных. — 2022. — URL: <https://studfile.net/preview/1504046/> (дата обращения 03.03.2023).
8. StudFile. Первая, вторая, третья нормальные формы. Нормальная форма Бойса-Кодда. — 2020. — URL: <https://studfile.net/preview/5917121/page:5/> (дата обращения 17.04.2023).
9. SQLzoo. SQLzoo. — 2019. — URL: https://sqlzoo.net/wiki/SQL_Tutorial (дата обращения 09.03.2023).
10. Советов Б. Я., Цехановский В. В., Чертовской В. Д. Базы данных. Теория и практика. — Москва : Юрайт, 2014. — 464 с.
11. PostgreSQLn. PostgreSQL: Documentation: 14: PostgreSQL 14.8 Documentation. — 2022. — URL: <https://www.postgresql.org/docs/14/index.html> (дата обращения 20.02.2023).
12. Postgrespro. Документация к Postgres Pro Standard 14. — 2016. — URL: <https://postgrespro.ru/docs/postgrespro/14/index> (дата обращения 24.02.2023).
13. PostgreSQLTutorial. PostgreSQL Tutorial – Comprehensive Postgresql Tutorial. — 2022. — URL: <https://www.postgresqltutorial.com/> (дата обращения 28.02.2023).

14. DockerDocs. Документация Docker. — 2021. — URL: <https://docs.docker.com/get-started/> (дата обращения 07.04.2023).
15. PostgresPro. PostgreSQL: Возможности шифрования. — 2022. — URL: <https://postgrespro.ru/docs/postgresql/14/encryption-options> (дата обращения 17.04.2023).
16. Stedihabr. Обеспечение безопасности базы данных PostgreSQL. — 2021. — URL: <https://habr.com/ru/articles/550882/> (дата обращения 11.04.2023).
17. 8host. Защита PostgreSQL от автоматизированных хакерских атак. — 2017. — URL: <https://www.8host.com/blog/zashhita-postgresql-ot-avtomatizirovannyx-xakerskix-atak/> (дата обращения 20.03.2023).
18. 8host. Тестирование производительности управляемой базы данных PostgreSQL с помощью pgbench. — 2019. — URL: <https://www.8host.com/blog/testirovanie-proizvoditelnosti-upravlyayemoj-bazy-dannyx-postgresql-s-pomoshhyu-pgbench/> (дата обращения 23.04.2023).
19. Chernigo L. Как ускорить работу PostgreSQL с помощью конфигурации базы и оптимизации запросов. — 2022. — URL: <https://habr.com/ru/companies/southbridge/articles/684826/> (дата обращения 01.05.2023).
20. Овсянкин А. Разбор оптимизаций запросов PostgreSQL на живых примерах. — 2019. — URL: <https://infostart.ru/1c/articles/1196217/> (дата обращения 30.04.2023).

ПРИЛОЖЕНИЕ А

Исходный код

На рисунке А.1 представлен QR-код с ссылкой на GitHub репозиторий со всеми файлами, ER-диаграммами, SQL-кодом разработанной базы данных и всего необходимого функционала.



Рисунок А.1 – Исходный код

ПРИЛОЖЕНИЕ Б

Нагрузочное тестирование

Для проведения нагрузочного тестирования было выбрано две конфигурации характеристик сервера БД.

Для начала продемонстрированы результаты выполнения с параметрами: CPU = 16, Memory = 32 GB, Hard Disk = 100 GB.

На рисунке Б.1 представлены результаты тестирования с параметрами «pgbench -i -s 10 database_name», где флаг -s 100 указывает на размер масштабируемости данных, где 10 означает, что будет создано в 10 раз больше строк, чем количество таблиц.

```
local@local-vm:~$ pgbench -i -s 10 -U postgres onmp
Password:
dropping old tables...
creating tables...
generating data (client-side)...
1000000 of 1000000 tuples (100%) done (elapsed 1.33 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 2.87 s (drop tables 0.65 s, create tables 0.00 s, client-side generate 1
.36 s, vacuum 0.39 s, primary keys 0.46 s).
```

Рисунок Б.1 – Тестирование с размером масштабируемости
данных 10

На рисунке Б.2 представлены результаты тестирования с параметрами «pgbench -i -s 100 database_name».

```
local@local-vm:~$ pgbench -i -s 100 -U postgres onmp
Password:
dropping old tables...
creating tables...
generating data (client-side)...
10000000 of 10000000 tuples (100%) done (elapsed 15.15 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 23.12 s (drop tables 0.01 s, create tables 0.01 s, client-side generate
15.27 s, vacuum 3.09 s, primary keys 4.74 s).
```

Рисунок Б.2 – Тестирование с размером масштабируемости
данных 100

На рисунке Б.3 представлены результаты тестирования с параметрами «pgbench -i -s 1000 database_name».

```
local@local-vm:~$ pgbench -i -s 1000 -U postgres onmp
Password:
dropping old tables...
creating tables...
generating data (client-side)...
1000000000 of 1000000000 tuples (100%) done (elapsed 157.57 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 288.18 s (drop tables 0.07 s, create tables 0.00 s, client-side generate
158.47 s, vacuum 64.52 s, primary keys 65.12 s).
```

Рисунок Б.3 – Тестирование с размером масштабируемости
данных 1000

На рисунке Б.4 представлены результаты тестирования с параметрами «pgbench -c 10 -j 10 -t 1000 database_name», где флаг -c 10 указывает на количество параллельных клиентов, -j 10 указывает на количество параллельных потоков, а -t 1000 указывает на общее количество транзакций для выполнения.

```
local@local-vm:~$ pgbench -c 10 -j 10 -t 1000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 10
number of threads: 10
number of transactions per client: 1000
number of transactions actually processed: 10000/10000
latency average = 1.469 ms
initial connection time = 61.481 ms
tps = 6806.985873 (without initial connection time)
```

Рисунок Б.4 – Тестирование с параметрами -c 10 -j 10 -t 1000

На рисунке Б.5 представлены результаты тестирования с параметрами «pgbench -c 10 -j 10 -t 10000 database_name».

```
local@local-vm:~$ pgbench -c 10 -j 10 -t 10000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 10
number of threads: 10
number of transactions per client: 10000
number of transactions actually processed: 100000/100000
latency average = 1.531 ms
initial connection time = 61.149 ms
tps = 6530.563199 (without initial connection time)
```

Рисунок Б.5 – Тестирование с параметрами -c 10 -j 10 -t 10000

На рисунке Б.6 представлены результаты тестирования с параметрами «pgbench -c 50 -j 50 -t 1000 database_name».

```
local@local-vm:~$ pgbench -c 50 -j 50 -t 1000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 50
number of threads: 50
number of transactions per client: 1000
number of transactions actually processed: 50000/50000
latency average = 4.233 ms
initial connection time = 253.849 ms
tps = 11810.779037 (without initial connection time)
```

Рисунок Б.6 – Тестирование с параметрами -c 50 -j 50 -t 1000

На рисунке Б.7 представлены результаты тестирования с параметрами «pgbench -c 50 -j 50 -t 10000 database_name».

```
local@local-vm:~$ pgbench -c 50 -j 50 -t 10000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 50
number of threads: 50
number of transactions per client: 10000
number of transactions actually processed: 500000/500000
latency average = 4.214 ms
initial connection time = 260.073 ms
tps = 11865.089002 (without initial connection time)
```

Рисунок Б.7 – Тестирование с параметрами -с 50 -ј 50 -т 10000

На рисунке Б.8 представлены результаты тестирования с параметрами «pgbench -с 80 -ј 80 -т 100000 database_name».

```
local@local-vm:~$ pgbench -c 80 -j 80 -t 100000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 80
number of threads: 80
number of transactions per client: 100000
number of transactions actually processed: 8000000/8000000
latency average = 5.587 ms
initial connection time = 401.133 ms
tps = 14317.816748 (without initial connection time)
```

Рисунок Б.8 – Тестирование с параметрами -с 80 -ј 80 -т 100000

Теперь изменим конфигурацию параметров системы: CPU = 32, Memory = 64 GB, Hard Disk = 200 GB.

На рисунке Б.9 представлены результаты тестирования с параметрами «pgbench -i -s 10 database_name», где флаг -s 100 указывает на размер масштабируемости данных, где 10 означает, что будет создано в 10 раз больше строк, чем количество таблиц.

```
local@local-vm:~$ pgbench -i -s 10 -U postgres onmp
Password:
dropping old tables...
creating tables...
generating data (client-side)...
1000000 of 1000000 tuples (100%) done (elapsed 1.45 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 2.46 s (drop tables 0.05 s, create tables 0.01 s, client-side generate 1
.49 s, vacuum 0.42 s, primary keys 0.48 s).
```

Рисунок Б.9 – Тестирование с размером масштабируемости
данных 10

На рисунке Б.10 представлены результаты тестирования с параметрами «pgbench -i -s 100 database_name».

```
local@local-vm:~$ pgbench -i -s 100 -U postgres onmp
Password:
dropping old tables...
creating tables...
generating data (client-side)...
10000000 of 10000000 tuples (100%) done (elapsed 15.56 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 23.69 s (drop tables 0.09 s, create tables 0.00 s, client-side generate
15.68 s, vacuum 2.81 s, primary keys 5.12 s).
```

Рисунок Б.10 – Тестирование с размером масштабируемости
данных 100

На рисунке Б.11 представлены результаты тестирования с параметрами «pgbench -i -s 1000 database_name».

```
local@local-vm:~$ pgbench -i -s 1000 -U postgres onmp
Password:
dropping old tables...
creating tables...
generating data (client-side)...
100000000 of 100000000 tuples (100%) done (elapsed 151.28 s, remaining 0.00 s)
vacuuming...
creating primary keys...
done in 567.64 s (drop tables 0.65 s, create tables 0.00 s, client-side generate
152.28 s, vacuum 135.58 s, primary keys 279.13 s).
```

Рисунок Б.11 – Тестирование с размером масштабируемости
данных 1000

На рисунке Б.12 представлены результаты тестирования с параметрами «pgbench -c 10 -j 10 -t 1000 database_name», где флаг -c 10 указывает на количество параллельных клиентов, -j 10 указывает на количество

параллельных потоков, а -t 1000 указывает на общее количество транзакций для выполнения.

```
local@local-vm:~$ pgbench -c 10 -j 10 -t 1000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 10
number of threads: 10
number of transactions per client: 1000
number of transactions actually processed: 10000/10000
latency average = 3.260 ms
initial connection time = 280.230 ms
tps = 3067.159130 (without initial connection time)
```

Рисунок Б.12 – Тестирование с параметрами -c 10 -j 10 -t 1000

На рисунке Б.13 представлены результаты тестирования с параметрами «pgbench -c 10 -j 10 -t 10000 database_name».

```
local@local-vm:~$ pgbench -c 10 -j 10 -t 10000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 10
number of threads: 10
number of transactions per client: 10000
number of transactions actually processed: 100000/100000
latency average = 2.014 ms
initial connection time = 93.045 ms
tps = 4965.758365 (without initial connection time)
```

Рисунок Б.13 – Тестирование с параметрами -c 10 -j 10 -t 10000

На рисунке Б.14 представлены результаты тестирования с параметрами «pgbench -c 50 -j 50 -t 1000 database_name».

```
local@local-vm:~$ pgbench -c 50 -j 50 -t 1000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 50
number of threads: 50
number of transactions per client: 1000
number of transactions actually processed: 50000/50000
latency average = 4.280 ms
initial connection time = 350.014 ms
tps = 11681.923648 (without initial connection time)
```

Рисунок Б.14 – Тестирование с параметрами -c 50 -j 50 -t 1000

На рисунке Б.15 представлены результаты тестирования с параметрами «pgbench -c 50 -j 50 -t 10000 database_name».

```
local@local-vm:~$ pgbench -c 50 -j 50 -t 10000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 50
number of threads: 50
number of transactions per client: 10000
number of transactions actually processed: 500000/500000
latency average = 3.889 ms
initial connection time = 360.088 ms
tps = 12856.418820 (without initial connection time)
```

Рисунок Б.15 – Тестирование с параметрами -c 50 -j 50 -t 10000

На рисунке Б.16 представлены результаты тестирования с параметрами «pgbench -c 80 -j 80 -t 10000 database_name».

```
local@local-vm:~$ pgbench -c 80 -j 80 -t 10000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 80
number of threads: 80
number of transactions per client: 10000
number of transactions actually processed: 8000000/8000000
latency average = 5.416 ms
initial connection time = 559.047 ms
tps = 14770.506577 (without initial connection time)
```

Рисунок Б.16 – Тестирование с параметрами -с 80 -ј 80 -т 10000

На рисунке Б.17 представлены результаты тестирования с параметрами «`pgbench -c 80 -j 80 -t 100000 database_name`».

```
local@local-vm:~$ pgbench -c 80 -j 80 -t 100000 -U postgres onmp
Password:
pgbench (14.7 (Ubuntu 14.7-0ubuntu0.22.04.1))
starting vacuum...end.
transaction type: <builtin: TPC-B (sort of)>
scaling factor: 1000
query mode: simple
number of clients: 80
number of threads: 80
number of transactions per client: 100000
number of transactions actually processed: 80000000/80000000
latency average = 6.675 ms
initial connection time = 629.178 ms
tps = 11985.601972 (without initial connection time)
```

Рисунок Б.17 – Тестирование с параметрами -с 80 -ј 80 -т 100000