

Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Модели данных и системы управления базами данных

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

**ПРОГРАММНОЕ СРЕДСТВО ДЛЯ ОРГАНИЗАЦИИ РАБОТЫ
МЕДИЦИНСКОГО ЦЕНТРА**

БГУИР КП 1-40 04 01 010 ПЗ

Студент

Д. А. Демидова

Руководитель

А. В. Давыдчик

Минск 2024

СОДЕРЖАНИЕ

Введение.....	4
1 Архитектура вычислительной системы.....	5
1.1 Структура и архитектура вычислительной системы.....	5
1.2 История, версии и достоинства.....	7
1.3 Обоснование выбора вычислительной системы.....	9
2 Платформа программного обеспечения.....	11
2.1 Выбор операционной системы.....	11
2.2 Выбор платформы для написания программы.....	12
3 Теоретическое обоснование разработки программного продукта.....	13
3.1 Обоснование необходимости разработки.....	13
3.2 Технологии программирования, используемые для решения.....	14
поставленных задач.....	14
4 Проектирование функциональных возможностей программы.....	16
4.1 Функциональные требования.....	16
4.2 Подключение к базе данных.....	16
4.3 Регистрация и авторизация пользователей.....	17
4.4 Функционал администратора.....	19
4.5 Функционал доктора.....	21
4.6 Функционал пациента.....	22
5 Проектирование разрабатываемой базы данных программного обеспечения.....	26
5.1 Разработка концептуальной модели.....	26
5.2 Разработка логической модели.....	28
5.3 Разработка физической модели.....	31
5.4 Конечная модель базы данных.....	32
5.5 Примеры запросов.....	32
Заключение.....	34
Список литературных источников.....	35

Приложение А (обязательное) Листинг программного кода.....	36
Приложение Б (обязательное) конечная схема базы данных.....	45
Приложение В (обязательное) ведомость курсового проекта.....	46

ВВЕДЕНИЕ

Современные медицинские учреждения ежедневно сталкиваются с необходимостью обработки большого объема данных, включая персональные данные пациентов, истории болезни, расписания врачей и другую информацию. Применение базы данных для хранения этих данных позволяет централизовать управление и организовать эффективное взаимодействие между сотрудниками медицинского центра. Правильно спроектированная система также позволяет соблюдать требования безопасности и конфиденциальности, которые являются важными аспектами работы с персональной информацией в медицинской сфере.

Целью данной курсовой работы является создание программного средства для организации работы медицинского центра, которое будет включать базу данных для хранения и управления информацией о пациентах, сотрудниках, услугах и медицинских записях. Создаваемое программное средство должно облегчить администрирование и ускорить доступ к медицинской информации, что особенно важно для обеспечения качественного и оперативного обслуживания пациентов.

Исходя из цели проекта был составлен следующий перечень задач:

- 1 Определить и обосновать перечень информационных сущностей для выбранной предметной области.
- 2 Разработать структуру базы данных, которая охватывает все необходимые аспекты работы медицинского центра, такие как управление пациентами, персоналом и медицинскими услугами.
- 3 Реализовать механизм взаимодействия с сущностями приложения.
- 4 Создать приложение, использующее разработанную базу данных.
- 5 Создать графический интерфейс для взаимодействия с приложением, использующим разработанную базу данных.

В ходе разработки программного средства будут использованы принципы проектирования и современные технологии для создания надежной и безопасной базы данных, которая станет важным инструментом для работы медицинского центра. В конечном итоге данная система должна упростить рабочие процессы и минимизировать ошибки, связанные с человеческим фактором, что будет способствовать повышению качества обслуживания пациентов.

1 АРХИТЕКТУРА ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

1.1 Структура и архитектура вычислительной системы

PostgreSQL – это объектно-реляционная система управления базами данных (ORDBMS), наиболее развитая из открытых СУБД в мире. Имеет открытый исходный код и является альтернативой коммерческим базам данных. С помощью PostgreSQL можно создавать, хранить базы данных и работать с данными с помощью запросов на языке SQL.

Одной из наиболее сильных сторон СУБД PostgreSQL является архитектура. Как и в случаях со многими коммерческими СУБД, PostgreSQL можно применять в среде клиент-сервер – это предоставляет множество преимуществ и пользователям, и разработчикам.

В основе PostgreSQL – серверный процесс базы данных, выполняемый на одном сервере. Доступ из приложений к данным базы PostgreSQL производится с помощью специального процесса базы данных. То есть клиентские программы не могут получать самостоятельный доступ к данным даже в том случае, если они функционируют на том же ПК, на котором осуществляется серверный процесс. пользователям, и разработчикам.

В основе PostgreSQL – серверный процесс базы данных, выполняемый на одном сервере. Доступ из приложений к данным базы PostgreSQL производится с помощью специального процесса базы данных. То есть клиентские программы не могут получать самостоятельный доступ к данным даже в том случае, если они функционируют на том же ПК, на котором осуществляется серверный процесс. [1]

Типичная модель распределенного приложения СУБД PostgreSQL (рисунок 1.1):

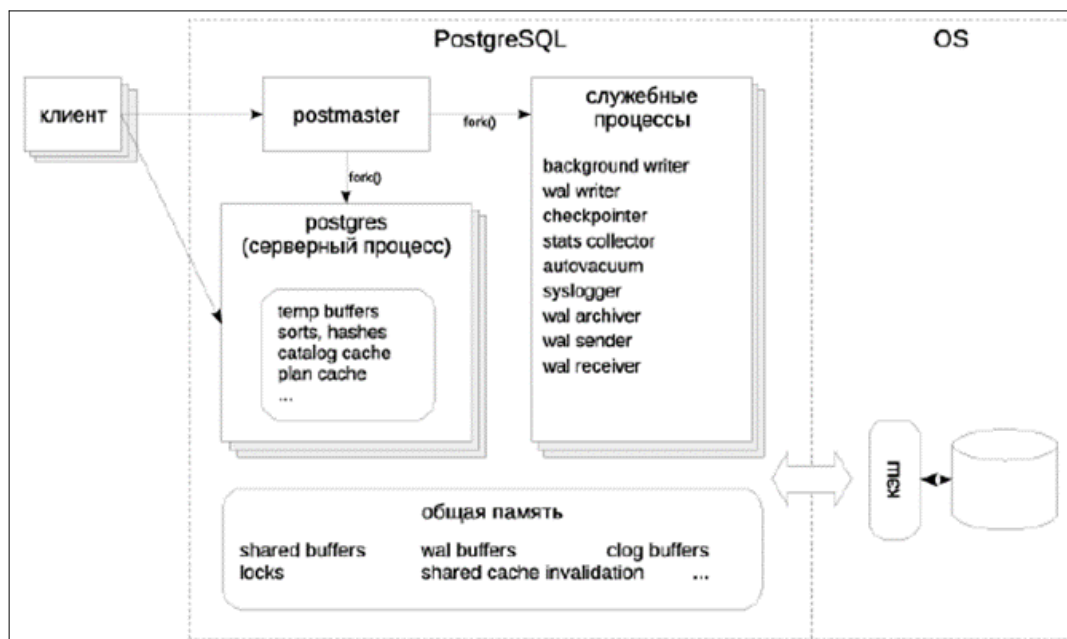


Рисунок 1.1 – Схема СУБД PostgreSQL

СУБД PostgreSQL ориентирована на протокол TCP/IP (локальная сеть либо Интернет), при этом каждый клиент соединён с главным серверным процессом БД (на рисунке 1.1 этот процесс обозначен Postmaster). Именно Postmaster создает новый серверный процесс специально в целях обслуживания запросов на доступ к данным определенного клиента. [2]

Сервер PostgreSQL может обрабатывать несколько одновременных подключений от клиентов. Для этого он запускает новый процесс для каждого соединения. С этого момента клиент и новый серверный процесс обмениваются данными без вмешательства исходного процесса postgres. Таким образом, процесс сервера-супервизора всегда работает, ожидая клиентских подключений, в то время как клиентские и связанные серверные процессы приходят и уходят.

Данные, которыми управляет PostgreSQL, хранятся в базах данных. Один экземпляр PostgreSQL одновременно работает с несколькими базами, которые вместе называются кластером баз данных.

Каталог, в котором размещаются все файлы, относящиеся к кластеру, обычно называют словом PGDATA, по имени переменной окружения, указывающей на этот каталог.

При инициализации в PGDATA создаются три одинаковые базы данных (рисунок 1.2):

1 template0 используется, например, для восстановления из логической резервной копии или для создания базы в другой кодировке и никогда не должна меняться;

2 template1 служит шаблоном для всех остальных баз данных, которые может создать пользователь в этом кластере;

3 postgres представляет собой обычную базу данных, которую можно использовать по своему усмотрению.

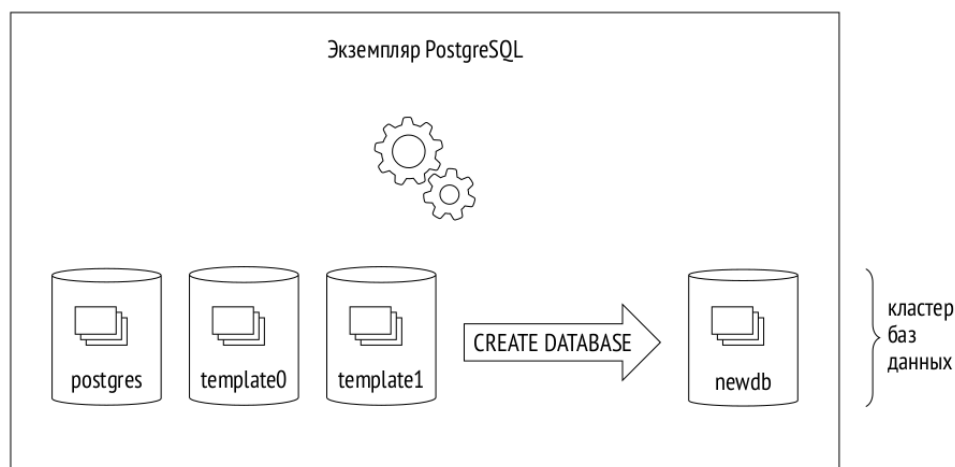


Рисунок 1.2 – Кластер PostgreSQL

Метаинформация обо всех объектах кластера (таких как таблицы, индексы, типы данных или функции) хранится в таблицах, относящихся к системному каталогу. В каждой базе данных имеется собственный набор таблиц (и представлений), описывающих объекты этой конкретной базы. Существует также несколько таблиц системного каталога, общих для всего кластера, которые не принадлежат какой-либо определенной базе данных и доступны в любой из них. [3]

1.2 История, версии и достоинства

Ранние версии системы были основаны на старой программе POSTGRES University, созданной университетом Беркли: так появилось название PostgreSQL. И сейчас СУБД иногда называют «Постгрес». Существуют сокращения PSQL и PgSQL – они тоже обозначают PostgreSQL.

По состоянию на июнь 2024 года PostgreSQL занимает четвертое место в общемировом рейтинге популярных СУБД (рисунок 1.3). [4]

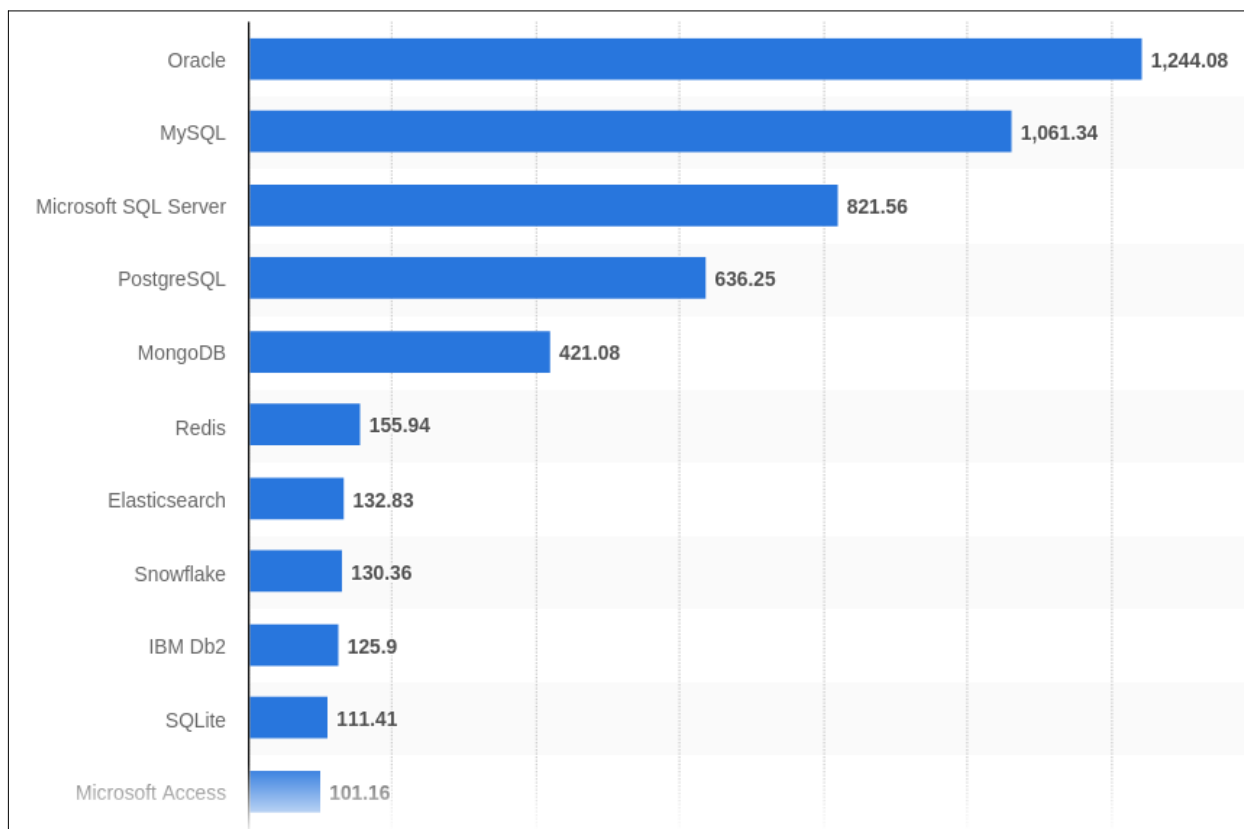


Рисунок 1.3 – Рейтинг популярности СУБД в июне 2024 года

У СУБД PostgreSQL много преимуществ, которые продолжают повышать ее популярность:

1 Любой специалист может бесплатно скачать, установить СУБД и сразу начать работу с базами данных.

2 PostgreSQL подходит для работы в любой операционной системе: Linux, macOS, Windows. Пользователь получает систему «из коробки» – чтобы установить и использовать программу, не нужны дополнительные инструменты.

3 PostgreSQL поддерживает много разных типов и структур данных, в том числе сетевые адреса, данные в текстовом формате JSON и геометрические данные для координат геопозиций. Все эти форматы можно хранить и обрабатывать в СУБД. Также при работе с PostgreSQL можно создавать собственные типы данных, их называют пользовательскими.

4 Размер базы данных в PostgreSQL не ограничен и зависит от того, сколько свободной памяти есть в месте хранения: на сервере, локальном компьютере или в облаке.

5 PostgreSQL реализует принципы ACID. Это четыре требования для надежной работы систем, которые обрабатывают данные в режиме реального

времени. Если все требования выполняются, данные не будут теряться из-за технических ошибок или сбоев в работе оборудования.

6 PostgreSQL поддерживает все современные функции баз данных: оконные функции, вложенные транзакции, триггеры.

7 Хотя большинство операций в PostgreSQL и используют классический стандарт языка SQL, помимо него поддерживается и свой отдельный диалект, позволяющий еще комфортнее писать запросы.

8 Поддерживается репликация «из коробки». Репликация – это сохранение копии базы данных. Копия может находиться на другом сервере.

9 PostgreSQL позволяет быстро без потерь перенести данные из другой СУБД. [5]

10 Возможность одновременного доступа к базе с нескольких устройств. В СУБД реализована клиент-серверная архитектура, когда база данных хранится на сервере, а доступ к ней осуществляется с клиентских компьютеров. Для ситуаций, когда несколько человек одновременно модифицируют базу используется технология MVCC – Multiversion Concurrency Control, многоверсионное управление параллельным доступом.

Благодаря перечисленным выше преимуществам иногда PostgreSQL называют бесплатным аналогом Oracle Database. Обе системы адаптированы под большие проекты и высокую нагрузку. Но есть разница: они по-разному хранят данные, предоставляют разные инструменты и различаются возможностями. Важная особенность PostgreSQL в том, что эта система – feature-rich: так называют проекты с широким функционалом. [6]

1.3 Обоснование выбора вычислительной системы

PostgreSQL выбрана для разработки приложения для медицинского центра, поскольку она сочетает в себе мощные функциональные возможности, надежность и высокую популярность среди реляционных СУБД. К июню 2024 года PostgreSQL занимает четвертое место в мировом рейтинге популярных СУБД, что подтверждает ее востребованность в сфере информационных технологий.

Одно из ключевых преимуществ PostgreSQL – это свободная лицензия, которая позволяет загружать и использовать систему без затрат. Она поддерживается на всех популярных операционных системах, включая Linux, macOS и Windows, что делает ее доступной для самых разных приложений и сред. Система предоставляет «из коробки» все необходимые инструменты,

позволяя сразу приступить к работе с базами данных, не требуя дополнительных программ или надстроек.

PostgreSQL выделяется поддержкой различных типов и структур данных, что особенно важно для приложений, которые обрабатывают сложные данные. В дополнение к стандартным типам данных она позволяет работать с JSON, геометрическими данными и сетевыми адресами. В данном проекте это облегчает хранение медицинской информации и расширяет возможности ее обработки.

Другой важный аспект – масштабируемость PostgreSQL. Размер базы данных ограничен лишь объемом доступной памяти на сервере или в облаке, что позволяет хранить большие объемы медицинских данных и обеспечивает длительное использование системы без необходимости миграции.

Надежность системы обусловлена ее соответствием стандартам ACID, что важно для приложений, которые требуют целостности и безопасности данных. Медицинские данные должны быть защищены от потерь и технических сбоев, и PostgreSQL предлагает стабильную платформу для их обработки и хранения.

Также PostgreSQL реализует многоверсионное управление параллельным доступом (MVCC), что позволяет нескольким пользователям одновременно работать с базой данных без блокировки доступа. Это ключевое преимущество для медицинского центра, где несколько сотрудников могут одновременно обращаться к базе для записи, просмотра и обновления данных пациентов.

Таким образом, PostgreSQL была выбрана для этого проекта благодаря ее широким функциональным возможностям, высокой надежности, поддержке разнообразных типов данных и оптимальной масштабируемости. Это делает PostgreSQL идеальной СУБД для медицинского центра, нуждающегося в эффективной и надежной системе управления данными.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

2.1 Выбор операционной системы

Linux – оптимальный выбор для приложения медицинского центра, поскольку данная операционная система обеспечивает надежную, безопасную и гибкую платформу для серверных решений, особенно тех, которые требуют высокой производительности и стабильности.

Linux известен своей высокой стабильностью, особенно на серверных платформах. В медицинском центре работа с данными пациентов и организация процесса требует круглосуточной доступности и минимального времени простоя. Linux стабильно работает даже при значительных нагрузках, что делает его идеальным для приложений с высоким объемом запросов и большим количеством данных.

Медицинские данные требуют особого уровня защиты, так как они содержат персональные данные и конфиденциальную информацию. Сильная встроенная система прав доступа (основанная на ролях) и возможность регулярных обновлений безопасности делают Linux безопасной ОС для хранения и управления конфиденциальными данными.

Также Linux позволяет оптимально использовать память и процессорные мощности, что особенно важно для приложений, обрабатывающих большое количество запросов и данных, как это требуется в медицинском центре. Благодаря тому, что Linux занимает минимум системных ресурсов, оставляя больше мощности для самого приложения и базы данных PostgreSQL, приложение будет работать быстрее и устойчивее.

Поскольку Linux является системой с открытым исходным кодом, она предоставляет экономически выгодное решение для медицинского центра. Установка и использование Linux не требуют затрат на лицензии, что снижает затраты на внедрение и эксплуатацию.

Кроме того, Linux предлагает множество инструментов для автоматизации задач: скрипты на Bash, планировщики задач (например, cron) и возможности контейнеризации (например, Docker). Эти инструменты можно использовать для регулярного резервного копирования базы данных, обновлений системы и настройки мониторинга. Возможность автоматизации поможет медицинскому центру обеспечить бесперебойную работу и уменьшить затраты на обслуживание системы.

Благодаря этим преимуществам, Linux является надежной и эффективной платформой для приложений медицинского центра, требующих высокой безопасности, производительности и стабильности.

Таким образом, в качестве операционной системы для проведения сравнения используется Linux (дистрибутив Ubuntu).

2.2 Выбор платформы для написания программы

В качестве языка программирования для написания программы используется Python. Python имеет несколько преимуществ для разработки приложения медицинского центра:

1 Python отличается простой и понятной синтаксической структурой, что сокращает время на написание и поддержку кода. Это позволяет разработчику быстрее перейти от идеи к работающему продукту, а также легко поддерживать и обновлять приложение.

2 Python предлагает широкий выбор библиотек и фреймворков для разработки серверных приложений, таких как Django и FastAPI. Эти фреймворки упрощают настройку и структуру серверной части, обеспечивая готовые решения для обработки запросов, аутентификации и работы с базами данных. Например, Django ORM позволяет упростить взаимодействие с PostgreSQL, делая его интуитивно понятным и надежным.

3 Python имеет обширные возможности для работы с базами данных. Библиотеки, такие как psycorg2 и SQLAlchemy, облегчают взаимодействие с PostgreSQL, позволяя выполнять как простые, так и сложные запросы к базе данных. Эти библиотеки предоставляют как интерфейс ORM, так и позволяют отправлять базы чистые запросы через так называемый «raw SQL».

В качестве платформы для разработки был выбран PyCharm. Главная причина этому то, что PyCharm является одной из наиболее популярных и мощных IDE для разработки на Python, предоставляя разработчикам множество инструментов и удобств для повышения производительности и качества кода, и в том числе для удобной интеграции с базой данных и отслеживания ее состояния.

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

3.1 Обоснование необходимости разработки

Эффективное управление данными является ключевым фактором успешной работы медицинских учреждений. Медицинские центры ежедневно сталкиваются с необходимостью обработки больших объемов информации: от записи пациентов на прием и хранения медицинской истории до планирования расписания врачей и управления финансовыми операциями. Без надлежащей системы автоматизации такие задачи требуют значительных временных и человеческих ресурсов, что может приводить к ошибкам, задержкам и снижению уровня обслуживания пациентов.

Создание программного средства для организации работы медицинского центра позволяет оптимизировать эти процессы и повысить общую эффективность работы учреждения. Программное обеспечение, разработанное на основе базы данных, дает возможность централизованного хранения, обработки и анализа данных. Это способствует более быстрому доступу к информации, снижает риск потери данных и минимизирует человеческий фактор при выполнении рутинных операций.

Кроме того, внедрение автоматизированной системы упрощает выполнение многих задач, включая управление расписанием врачей, обработку заявок от пациентов, формирование отчетов для руководства и мониторинг занятости ресурсов. Такие функции особенно актуальны в условиях высокой нагрузки, когда ручное управление данными становится непрактичным.

Необходимость разработки подобного программного обеспечения обусловлена и современными требованиями в области медицины, включая соблюдение стандартов безопасности данных пациентов. Программное средство, основанное на современных СУБД, таких как PostgreSQL, позволяет обеспечить высокий уровень надежности, целостности и защиты данных.

Важно отметить, что разработка такого программного средства не только упрощает управление данными, но и способствует улучшению качества предоставляемых услуг. Пациенты получают возможность оперативного взаимодействия с медицинским центром, будь то запись на прием или получение медицинской документации. Руководство же получает

мощный инструмент для анализа ключевых показателей работы учреждения и принятия обоснованных решений.

Таким образом, разработка программного средства для организации работы медицинского центра является важным шагом к повышению эффективности управления, улучшению качества обслуживания пациентов и внедрению современных технологий в медицинскую сферу. Такое решение отвечает как текущим потребностям медицинских центров, так и тенденциям их цифровой трансформации.

3.2 Технологии программирования, используемые для решения поставленных задач

Для решения задач курсового проекта был сделан выбор в пользу минималистичного набора технологий. Основной акцент был сделан на глубокой интеграции с базой данных, чтобы максимально эффективно управлять данными, сохраняя контроль за выполнением операций на уровне SQL-запросов.

Одной из ключевых библиотек, использованных в разработке, является `psycopg2` – официальная библиотека для работы с PostgreSQL в Python. Она предоставляет удобный и гибкий интерфейс для выполнения SQL-запросов, управления транзакциями, а также работы с курсорами. Такой подход к реализации позволяет разработчику напрямую взаимодействовать с базой данных, избегая промежуточных абстракций, которые могли бы усложнить контроль за процессом выполнения операций. Например, все транзакции в программе (коммиты и откаты) управляются вручную, что гарантирует предсказуемое и безопасное выполнение операций с данными. [7]

Для создания графического пользовательского интерфейса было выбрано средство `tkinter`. Это встроенная в Python библиотека, которая позволяет разрабатывать удобные и функциональные интерфейсы без необходимости использования сторонних инструментов. Ее использование обеспечивает простоту интеграции с основной логикой программы и позволяет создать интерфейс, полностью адаптированный под поставленные задачи. [8]

Дополнительно, для упрощения работы с базой данных в процессе разработки был использован инструмент `Docker`. Это решение позволило избежать необходимости установки PostgreSQL на локальной машине разработчика, что делает разработку более гибкой и удобной. Использование `Docker` обеспечивает возможность быстрого развертывания базы данных в

изолированном контейнере, позволяя минимизировать риски конфликтов зависимостей и упрощая настройку среды разработки. Более того, данный подход легко масштабируется и обеспечивает удобство переноса проекта на другие машины или серверы, что делает его особенно актуальным для проектов с использованием реляционных баз данных. [9]

Несмотря на относительно небольшой набор используемых библиотек и инструментов, это решение было осознанным и связано с целью сосредоточить внимание на работе с базой данных через сырой SQL. Такой подход позволяет полностью контролировать процесс взаимодействия с данными, учитывать особенности выполнения транзакций и корректно обрабатывать возможные ошибки. В отличие от ORM (Object-Relational Mapping), данный подход исключает дополнительные абстракции и дает гибкость в использовании возможностей PostgreSQL. [10]

Таким образом, выбранный стек технологий соответствует задачам курсовой работы, обеспечивая не только необходимую функциональность, но и высокую степень контроля за всеми этапами работы с базой данных. На основе этого можно сделать вывод о том, что используемые технологии программирования позволяют выполнить цели данного курсового проекта.

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

4.1 Функциональные требования

Программа должна иметь систему ролей, состоящую из администратора, доктора и пациента, так как их пользовательский опыт в рамках медицинского центра отличается.

Администратор должен быть способен осуществлять CRUD-операции над всеми сущностями программы, иметь доступ ко всем данным.

Пациент, чтобы успешно пользоваться врачебными услугами, должен иметь возможность:

- 1 Просмотреть список своих записей к докторам;
- 2 Записаться к доктору на свободное окошко и выбрать предоставляемую услугу;
- 3 Просмотреть список всех когда-либо выписанных ему диагнозов (аналогично медицинской книжке в поликлинике);
- 4 Просмотреть список всех когда-либо выданных предписаний от докторов.

Доктор, чтобы успешно оказывать врачебные услуги, должен иметь возможность:

- 1 Просмотреть список записанных к нему пациентов;
- 2 Просматривать детальную информацию о посещении, такую как тип оказываемой услуги, дату и время приема;
- 3 По итогам проведенного приема добавлять в медицинскую книжку пациента какой-либо диагноз, а также врачебное предписание;
- 4 Помечать прием, как закрытый.

Кроме того, система конечно же должна предоставлять пользователям возможность зарегистрироваться, заполнить свой профиль, а также заходить в приложение по логину и паролю.

4.2 Подключение к базе данных

Для более экономного расходования ресурсов компьютера база данных поднимается не локально, а в Docker. Это не только избавляет пользователей от необходимости тратить время на установку базы данных локально и ее конфигурацию, но также занимает гораздо меньше памяти, так как был

подобран Docker-образ (Docker-image), имеющий оптимальный баланс между производительностью и размером – Postgres 16 в версии Alpine, который весит всего 275 мегабайт. Запускается такая база данных одной простой командой «docker compose up».

На уровне приложения подключение к базе данных осуществляется с помощью библиотеки psycopg2. В приложении есть отдельный класс для работы с базой данных Database, который хранит подключение к базе в одном из своих полей. Именно этот класс используется во всех контроллерах приложения, что позволяет разделить код, связанный с взаимодействием с базой данных, и код, обрабатывающий действия пользователя. Кроме того, если база данных еще не настроена (например в случае, если это первый запуск приложения), то данный класс вызовет функции create_database и create_tables, которые запустят нужные SQL-скрипты для настройки базы данных.

Также данный класс в стиле raw-SQL выполняет все запросы к базе данных, а также контролирует фиксацию (commit) и откат (rollback) изменений.

4.3 Регистрация и авторизация пользователей

При входе в приложение пользователя встречает окно регистрации и входа (рисунок 4.1):

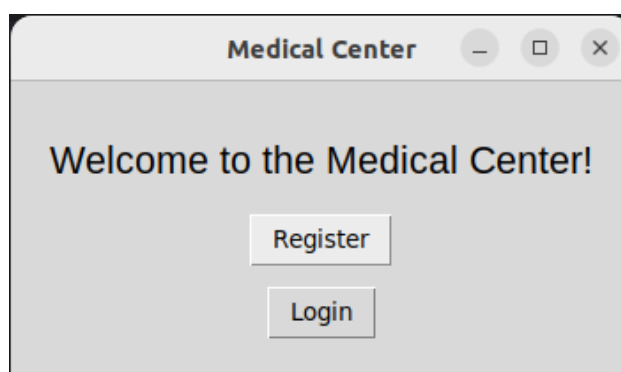


Рисунок 4.1 – Окно регистрации и входа

Для входа в систему достаточно ввести логин и пароль (рисунок 4.2):

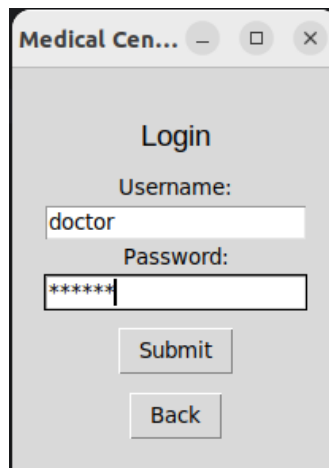


Рисунок 4.2 – Окно входа

После входа пользователя ждет приветствие (рисунок 4.3):

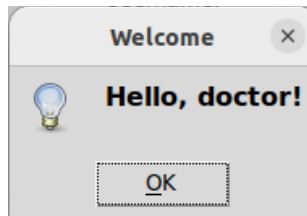


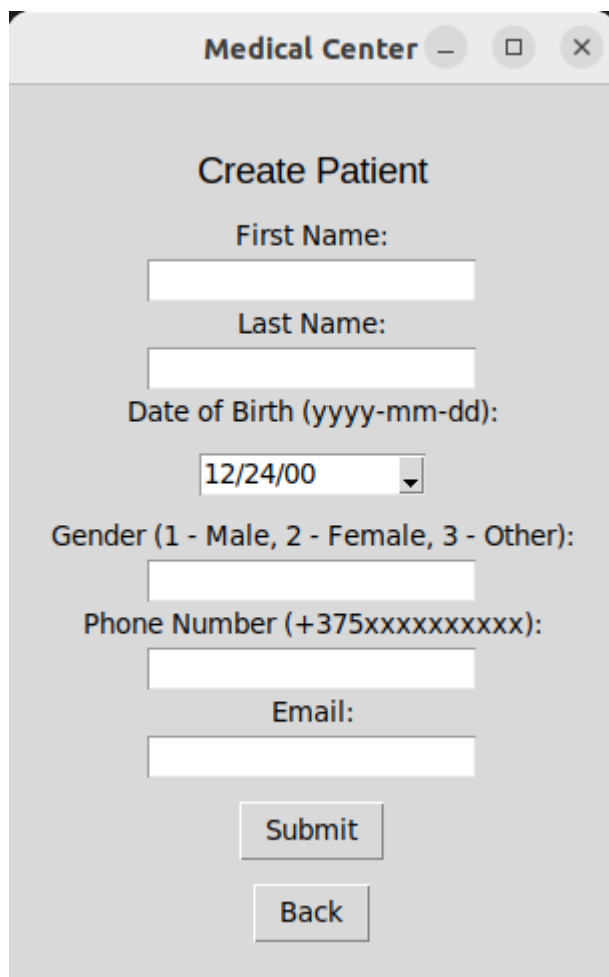
Рисунок 4.3 – Приветствие пользователя

При регистрации пользователю нужно придумать себе логин и пароль, а также выбрать свою роль (пациент, доктор) (рисунок 4.4):



Рисунок 4.4 – Регистрация пользователя

После регистрации пользователь должен заполнить данные о себе (рисунок 4.5):



The screenshot shows a web application window titled "Medical Center". Inside, there is a form titled "Create Patient". The form contains the following fields and controls:

- First Name:** A text input field.
- Last Name:** A text input field.
- Date of Birth (yyyy-mm-dd):** A date input field with a dropdown arrow, currently showing "12/24/00".
- Gender (1 - Male, 2 - Female, 3 - Other):** A text input field.
- Phone Number (+375xxxxxxxxxx):** A text input field.
- Email:** A text input field.
- Submit:** A button.
- Back:** A button.

Рисунок 4.5 – Заполнение профиля пользователя

После завершения регистрации пользователь может полноценно использовать приложение.

Стоит также отметить, что создание администратора через графический интерфейс не предусмотрено – нового администратора может создать напрямую в базе данных человек, имеющий к ней доступ, – другой администратор. Всем остальным функционалом (входом и изменением сущностей) администраторы могут пользоваться и через интерфейс.

4.4 Функционал администратора

В приложении администратор имеет полную власть и может манипулировать всеми сущностями приложения, поэтому при входе в

систему администратору ему предлагается выбрать, с какой сущностью он хочет работать (рисунок 4.6):

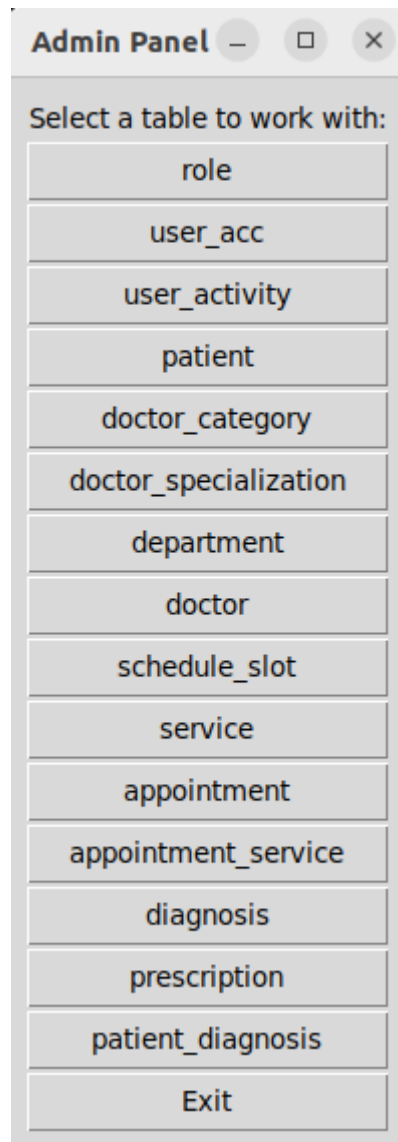


Рисунок 4.6 – Выбор сущности для работы

Выбрав сущность для работы администратору предоставляется возможный набор операций над сущностью: показать весь список, добавить новую единицу, изменить сущность, удалить сущность, показать определенную сущность (рисунок 4.7):

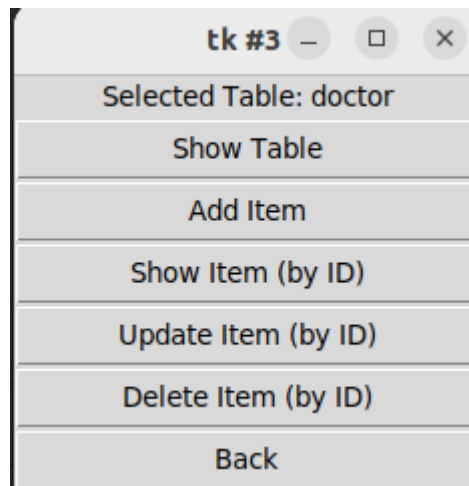


Рисунок 4.7 – Варианты работы с таблицей

Таким образом, интерфейс администратора полностью реализует желаемый функционал.

4.5 Функционал доктора

Доктор может просматривать свои приемы, а также отдельно взаимодействовать с ними (рисунок 4.8):

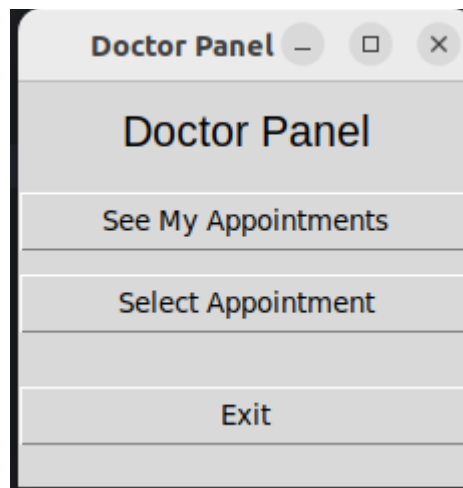


Рисунок 4.8 – Опции доктора

При просмотре приемов видны имена пациентов, услуги, а также время и дата приема (рисунок 4.9):

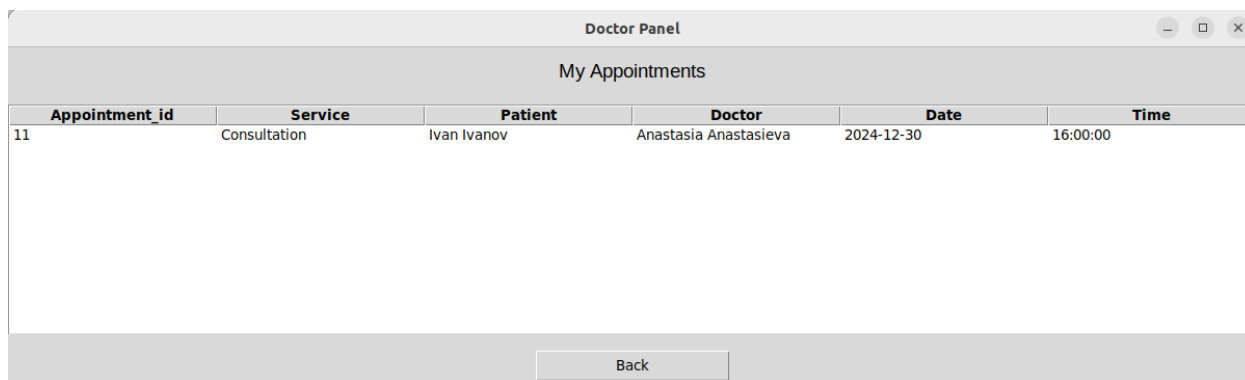


Рисунок 4.9 – Просмотр приемов доктора

Доктор также может выбрать конкретный прием из списка и работать с ним. Доктор может просмотреть детали приема, выписать пациенту диагноз и предписание, а также пометить прием завершенным (рисунок 4.10):

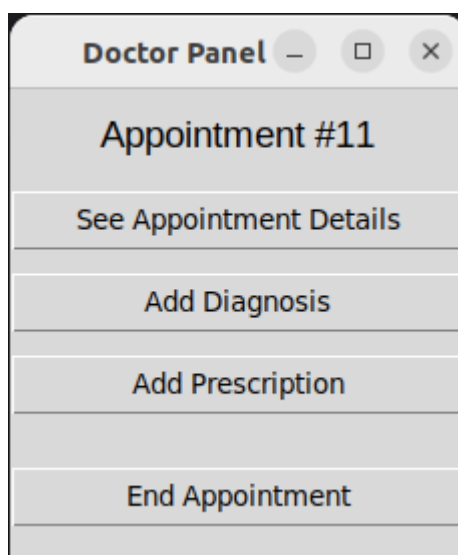


Рисунок 4.10 – Опции работы с приемом

Таким образом, интерфейс доктора полностью реализует желаемый функционал.

4.6 Функционал пациента

Пациент может просматривать свои приемы, записываться на приемы, а также просматривать свою историю диагнозов и предписаний (рисунок 4.11):

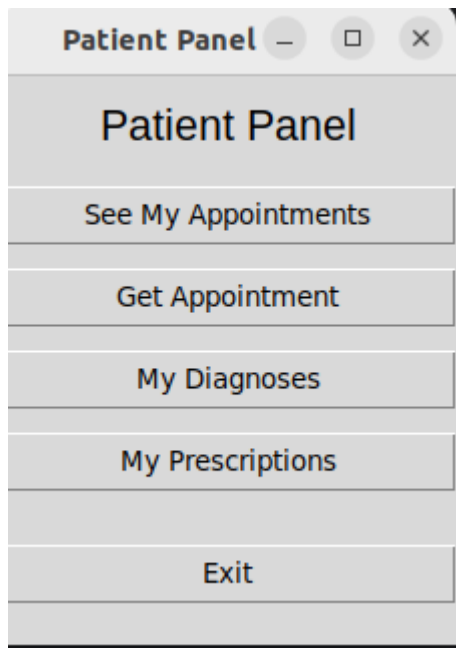


Рисунок 4.11 – Опции пациента

При просмотре приемов видны имена докторов, услуги, а также время и дата приема (рисунок 4.12):

Doctor Panel

My Appointments

Appointment_id	Service	Patient	Doctor	Date	Time
11	Consultation	Ivan Ivanov	Anastasia Anastasieva	2024-12-30	16:00:00

Back

Рисунок 4.12 – Просмотр приемов пациента

Пациент также может записаться на прием при наличии свободных окошек (рисунок 4.13):

Patient Panel

Get Appointment

Id	Date	Time	Doctor
12	2024-12-24	20:00:00	Ivan Ivanov
13	2024-12-24	19:00:00	Ivan Ivanov

Select Slot ID:

Next

Back

Рисунок 4.13 – Запись на прием

После выбора свободного окошка пользователь также может выбрать услугу данного врача из его списка услуг (рисунок 4.14):

Patient Panel

Select Service

Service id	Service name	Price
1	Consultation	200.00

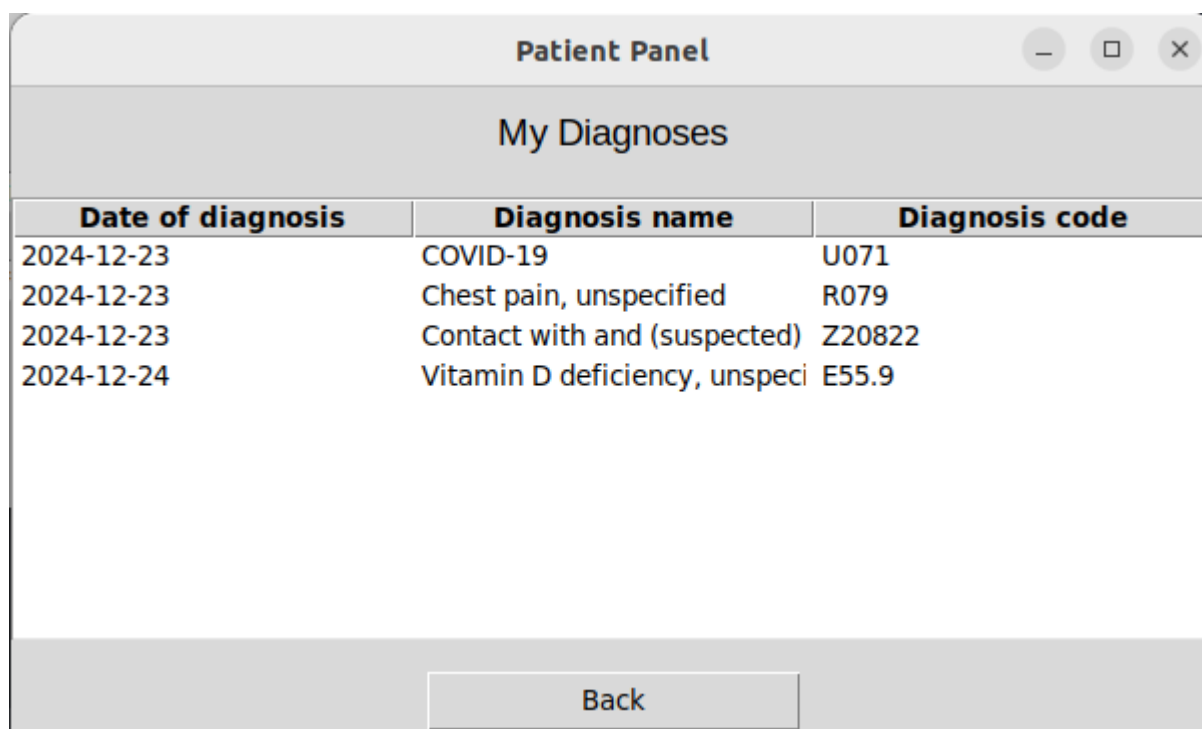
Select Service ID:

Confirm

Back

Рисунок 4.14 – Выбор услуги на прием

Также пациент может просмотреть историю своих диагнозов (рисунок 4.15) и предписаний.



The screenshot shows a window titled "Patient Panel" with standard window controls (minimize, maximize, close) in the top right corner. Below the title bar is a header section labeled "My Diagnoses". Underneath this header is a table with three columns: "Date of diagnosis", "Diagnosis name", and "Diagnosis code". The table contains four rows of data. At the bottom of the window, there is a "Back" button.

Date of diagnosis	Diagnosis name	Diagnosis code
2024-12-23	COVID-19	U071
2024-12-23	Chest pain, unspecified	R079
2024-12-23	Contact with and (suspected)	Z20822
2024-12-24	Vitamin D deficiency, unspeci	E55.9

Рисунок 4.15 – Просмотр истории диагнозов

Таким образом, спроектированное приложение полностью соответствует ранее сформулированным функциональным требованиям.

5 ПРОЕКТИРОВАНИЕ РАЗРАБАТЫВАЕМОЙ БАЗЫ ДАННЫХ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Проектирование базы данных является важным этапом в создании программного обеспечения, обеспечивающим эффективность хранения, обработки и использования данных. Для создания хорошо структурированной базы данных необходимо следовать поэтапному процессу. Сначала разрабатывается концептуальная модель, которая позволяет выделить ключевые сущности системы и их взаимосвязи. Затем создается логическая модель, где уточняются атрибуты сущностей и устанавливаются ограничения для поддержания целостности данных. На следующем этапе проектируется физическая модель, в которой происходит реализация структуры базы данных на уровне СУБД, включая типы данных и индексы. Завершается процесс формированием окончательной модели, полностью готовой к использованию в приложении. Такой системный подход к проектированию базы данных гарантирует, что система будет соответствовать функциональным требованиям и обеспечивать стабильную работу.

5.1 Разработка концептуальной модели

На этом этапе необходимо выделить основные сущности системы, определить их свойства и взаимосвязи. Концептуальная модель представляет собой абстрактное представление данных, независимое от конкретной платформы или технологии. Она позволяет лучше понять структуру и функциональность будущей базы данных, а также убедиться в том, что она будет отвечать всем требованиям, предъявляемым к системе.

В данном проекте для медицинского центра ключевыми сущностями являются:

1 Пользователи (User), которые делятся на три основные роли: администраторы, доктора и пациенты. У каждого пользователя есть уникальное имя, пароль и привязанная роль.

2 Пациенты (Patient) должны иметь возможность записываться к врачам, просматривать свои диагнозы, предписания и записи на прием. Для пациента хранятся персональные данные, такие как имя, фамилия, дата рождения, пол, контактные данные.

3 Доктора (Doctor), с привязкой к специальности и отделению. Для каждого доктора фиксируются имя, фамилия, пол, а также категории и специализации, которые структурируются в отдельных сущностях.

4 Услуги (Service), предоставляемые докторами. Услуга включает название, стоимость и связь с конкретным врачом.

5 Расписание (Schedule Slot), где фиксируются доступные временные слоты для записи к доктору. Каждый слот может быть отмечен как свободный или занятый.

6 Записи на прием (Appointment), связывающие пациента, временной слот и предоставленные услуги.

7 Диагнозы (Diagnosis) и предписания (Prescription), которые назначаются докторами в ходе приема и сохраняются для пациентов.

Разработка концептуальной модели позволяет сосредоточиться на реальных бизнес-процессах и представить данные в виде связанных сущностей. Это важно для дальнейших этапов проектирования, так как на основе концептуальной модели создается логическая структура базы данных.

Концептуальная схема представлена на рисунке 5.1.

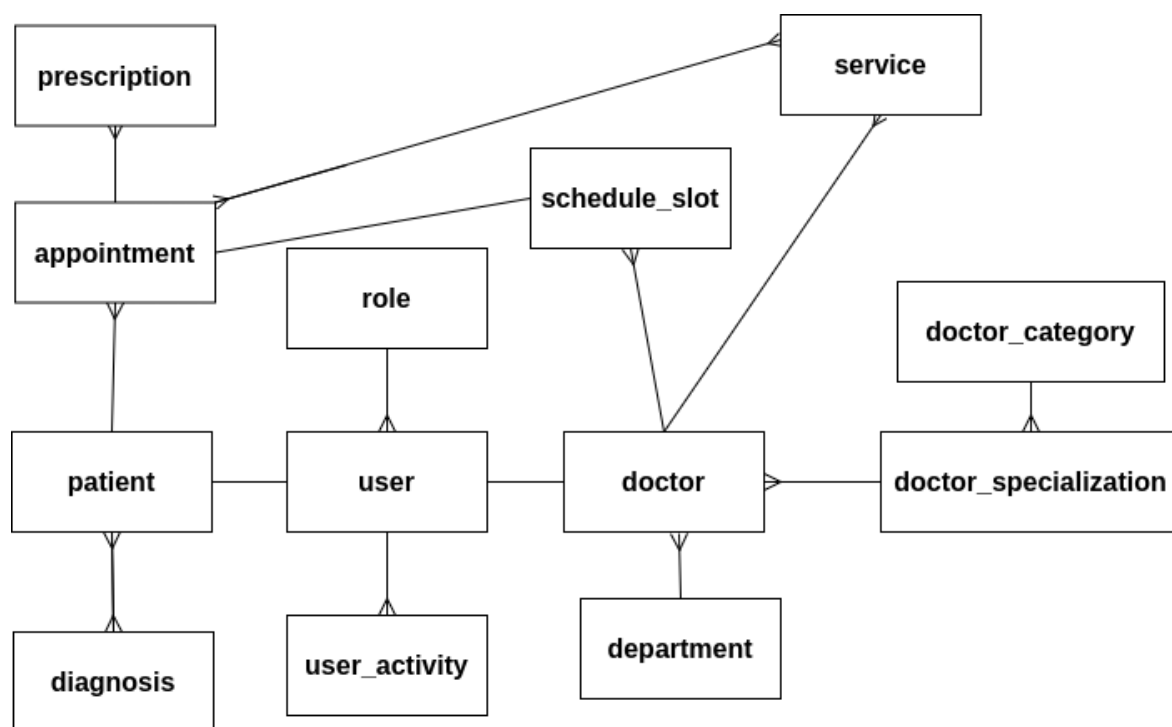


Рисунок 5.1 – Схема концептуальной модели базы данных

Данная схема позволяет упростить следующий этап проектирования – создание логической модели базы данных.

5.2 Разработка логической модели

После создания концептуальной модели следующим шагом является разработка логической модели базы данных. Логическая модель уточняет структуру данных, добавляя информацию о типах данных, первичных и внешних ключах, а также связях между таблицами. Этот этап приближает модель к технической реализации, но остается независимым от конкретной платформы базы данных.

В логической модели для приложения медицинского центра важным шагом стало определение связей между сущностями. Например:

1 Таблица «User» содержит базовую информацию о пользователях и связана с таблицей «Role», которая определяет тип пользователя: администратор, доктор или пациент. Это упрощает управление ролями и обеспечивает их уникальность.

2 «Doctor» и «Patient» являются специализированными сущностями, которые включают дополнительную информацию о врачах и пациентах. Эти таблицы имеют связь «один к одному» с таблицей «User», чтобы минимизировать дублирование данных.

3 Для расписания доктора используется таблица «Schedule slot», где хранятся дата и время слота, а также информация о том, занят он или нет. Связь с таблицей «Doctor» помогает структурировать доступность врачей по времени.

4 Таблицы «Appointment» и «Appointment service» поддерживают связи между пациентами, расписанием и услугами. Это позволяет гибко описывать приемы, включающие несколько услуг.

5 Таблицы «Diagnosis» и «Prescription» дают возможность сохранять данные о диагнозах и предписаниях. Модель поддерживает связь «многие ко многим» через таблицу «Patient diagnosis», чтобы пациент мог иметь несколько диагнозов.

Логическая модель также учитывает ограничения целостности данных:

1 Первичные ключи устанавливаются для каждой таблицы, например, «id» в таблицах «User», «Doctor», «Patient».

2 Внешние ключи определяют связи между таблицами, например, в «Appointment» поля «slot_id» и «patient_id» ссылаются на соответствующие таблицы.

3 Ограничения уникальности гарантируют, что поля, такие как «username» в «User» или «specialization_name» в «Doctor specialization», остаются уникальными.

Данная схема позволяет упростить следующий этап проектирования – создание физической модели базы данных.

5.3 Разработка физической модели

Физическая модель базы данных включает в себя преобразование логической модели в конкретную реализацию с использованием языка SQL. На этом этапе учитываются особенности выбранной системы управления базами данных (в данном случае PostgreSQL), а также применяются оптимизации для повышения производительности и удобства работы.

Физическая модель основывается на логической модели, разработанной ранее. Для ее реализации создаются SQL-скрипты, которые:

1 Определяют структуру таблиц. Каждая сущность из логической модели преобразуется в таблицу с соответствующими типами данных и ограничениями. Например, для идентификаторов «id» используется тип данных «SERIAL» или «BIGSERIAL», чтобы обеспечить автоинкремент значений. Поля, содержащие строки, такие как «username» или «email», используют типы «VARCHAR» или «TEXT» с ограничениями уникальности, если это необходимо.

2 Устанавливают связи между таблицами. Добавляются внешние ключи («FOREIGN KEY»), обеспечивающие целостность данных. Например, таблица «Doctor» ссылается на таблицы «User», «Department» и «Doctor specialization» для связывания информации о враче с основной учетной записью и дополнительными данными. Таблица «Appointment» включает внешние ключи для связи с таблицами «Schedule slot», «Patient» и «Prescription».

3 Определяют индексы. Для ускорения операций чтения создаются индексы на полях, которые часто используются в запросах. Например, индексы на поле «username» в таблице «User» или на поле «email» в таблице «Patient».

4 Учитывают особенности PostgreSQL. Например, регулярные выражения применяются для проверки форматов данных, таких как номер телефона или адрес электронной почты. В случае сложных связей используются промежуточные таблицы для реализации отношений «многие ко многим», как, например, «Appointment service» и «Patient diagnosis».

Физическая модель в виде полного SQL-скрипта представлена в Приложении А. Она включает команды создания таблиц, установки внешних ключей, ограничений целостности, а также начальное наполнение данных.

5.4 Конечная модель базы данных

Конечная модель базы данных представляет собой заверченный этап проектирования, в результате которого система полностью готова к использованию. Она включает все элементы, определенные на этапе физической модели, и обеспечивает выполнение функциональных требований, поставленных в начале разработки.

Эта модель соединяет в себе структурную целостность, производительность и удобство эксплуатации. Все таблицы, их атрибуты и связи были реализованы таким образом, чтобы полностью соответствовать логической модели и учитывать особенности предметной области.

Конечная модель включает следующие ключевые компоненты:

1 Таблицы для хранения данных о пользователях и их ролях: «User», «Role», «User activity».

2 Таблицы для представления медицинской информации, такие как «Patient», «Doctor», «Appointment», «Diagnosis», «Prescription» и другие.

3 Промежуточные таблицы для обработки сложных связей, такие как «Appointment service» и «Patient diagnosis».

4 Поля с ограничениями уникальности и проверками форматов, что обеспечивает высокий уровень достоверности данных.

На этом этапе учитывались требования к производительности, поэтому были добавлены индексы на наиболее часто используемые поля, такие как «username», «email», «doctor_id» и другие. Это позволяет значительно ускорить выполнение запросов, что особенно важно для приложения, работающего с большими объемами данных.

Схема конечной модели базы данных представлена в Приложении Б. Она наглядно демонстрирует структуру таблиц, их связи и ключевые атрибуты, обеспечивая понимание архитектуры базы данных.

5.5 Примеры запросов

Понимание возможностей базы данных невозможно без практической демонстрации ее работы. Поэтому ниже приведены примеры SQL-запросов к спроектированной базе данных, чтобы оценить, насколько корректно и эффективно реализованы структуры и связи, заложенные в проекте.

Примеры запросов к спроектированной базе данных приведены в Приложении А в файлах queries.sql и complex_queries.sql.

Таким образом, спроектированная база данных обеспечивает выполнение всех поставленных целей и функциональных требований, необходимых для организации работы медицинского центра. На каждом этапе проектирования были учтены ключевые аспекты функциональности, включая управление ролями пользователей, обработку записей пациентов, назначение услуг и управление расписанием докторов.

Результатом работы стала структурированная, надежная и гибкая база данных, которая поддерживает все заявленные функции: выполнение CRUD-операций и эффективное взаимодействие между различными сущностями системы. Реализация всех этапов проектирования позволила создать модель данных, отвечающую как техническим, так и бизнес-требованиям приложения.

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта была спроектирована и реализована база данных для программного обеспечения, предназначенного для организации работы медицинского центра. Были изучены функциональные требования системы, разработаны концептуальная, логическая и физическая модели базы данных. На основе разработанных моделей создана структура базы данных, реализованная с использованием СУБД PostgreSQL и библиотеки psycopg2. Для удобства работы с данными был также разработан графический интерфейс пользователя с применением библиотеки Tkinter.

В процессе работы особое внимание уделялось точности проектирования реляционной структуры, включая определение ключевых сущностей, их атрибутов и связей. Это позволило учесть особенности работы медицинского центра, такие как поддержка ролей пользователей (администратор, доктор, пациент), управление записями пациентов, назначение услуг, обработка расписания докторов.

Таким образом, результатом выполнения данной курсовой работы стала функциональная и гибкая база данных, которая удовлетворяет заявленным функциональным требованиям и обеспечивает удобство использования. Проект продемонстрировал практическое применение современных технологий программирования и методов проектирования баз данных, а также подтвердил эффективность ручной работы с SQL-запросами.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] СУБД PostgreSQL. Особенности и архитектура Postgres | OTUS [Электронный ресурс]. – Режим доступа: <https://otus.ru/nest/post/1584/> – Дата доступа: 01.10.2024.

[2] Изучаем PostgreSQL. Часть 1. Знакомимся с архитектурой / Хабр [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/companies/otus/articles/706346/> – Дата доступа: 01.10.2024.

[3] Рогов Е. В. PostgreSQL 16 изнутри. М.: ДМК Пресс, 2024. 664 с.

[4] Most popular database management systems 2024 | Statista [Электронный ресурс]. – Режим доступа: <https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/> – Дата доступа: 02.10.2024.

[5] PostgreSQL: что это, для чего нужна, основы и преимущества - установка и настройка СУБД Postgres [Электронный ресурс]. – Режим доступа: <https://practicum.yandex.ru/blog/chto-takoe-subd-postgresql/> – Дата доступа: 03.10.2024.

[6] PostgreSQL: что это за СУБД, основы и преимущества [Электронный ресурс]. – Режим доступа: <https://blog.skillfactory.ru/glossary/postgresql/> – Дата доступа: 04.10.2024.

[7] Psycopg – PostgreSQL database adapter for Python — Psycopg 2.9.10 documentation [Электронный ресурс]. – Режим доступа: <https://www.psycopg.org/docs/> – Дата доступа: 05.11.2024.

[8] Graphical User Interfaces with Tk — Python 3.13.1 documentation [Электронный ресурс]. – Режим доступа: <https://docs.python.org/3/library/tk.html> – Дата доступа: 15.11.2024.

[9] Docker Docs [Электронный ресурс]. – Режим доступа: <https://docs.docker.com/> – Дата доступа: 20.11.2024.

[10] The pros and cons of using raw SQL versus ORM for database development | by Ritika adequate | Medium [Электронный ресурс]. – Режим доступа: <https://medium.com/@ritika.adequate/the-pros-and-cons-of-using-raw-sql-versus-orm-for-database-development-e9edde7ee31e> – Дата доступа: 01.12.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

Файл create_tables.sql.

```
CREATE TABLE IF NOT EXISTS role (  
    id SERIAL PRIMARY KEY NOT NULL,  
    role_name VARCHAR(25) UNIQUE NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS user_acc (  
    id BIGSERIAL PRIMARY KEY NOT NULL,  
    username VARCHAR(50) UNIQUE NOT NULL,  
    password VARCHAR(50) NOT NULL,  
    role_id INT REFERENCES role (id) ON DELETE CASCADE NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS user_activity (  
    id BIGSERIAL PRIMARY KEY NOT NULL,  
    user_id BIGINT REFERENCES user_acc (id) ON DELETE CASCADE NOT NULL,  
    activity_type VARCHAR(255) NOT NULL,  
    date_of_activity DATE NOT NULL DEFAULT CURRENT_DATE,  
    time_of_activity TIME NOT NULL DEFAULT CURRENT_TIME  
);  
  
CREATE TABLE IF NOT EXISTS patient (  
    id BIGSERIAL PRIMARY KEY NOT NULL,  
    user_id BIGINT REFERENCES user_acc (id) ON DELETE CASCADE UNIQUE NOT  
NULL,  
    first_name VARCHAR(50) NOT NULL,  
    last_name VARCHAR(50) NOT NULL,  
    date_of_birth DATE NOT NULL,  
    gender VARCHAR(20) NOT NULL,  
    phone_number CHAR(13) NOT NULL,  
    email VARCHAR(50),  
    CONSTRAINT proper_email CHECK (email ~* '.*@.+\\.\\.\\.\\.*'),  
    CONSTRAINT proper_phone CHECK (phone_number ~* '\\+375[0-9]{9}'))  
);  
  
CREATE TABLE IF NOT EXISTS doctor_category (  
    id SERIAL PRIMARY KEY NOT NULL,  
    category_name VARCHAR(100) UNIQUE NOT NULL  
);  
  
CREATE TABLE IF NOT EXISTS doctor_specialization (  
    id SERIAL PRIMARY KEY NOT NULL,  
    specialization_name VARCHAR(100) UNIQUE NOT NULL,  
    category_id INT REFERENCES doctor_category (id) ON DELETE SET NULL  
);
```

```

CREATE TABLE IF NOT EXISTS department (
    id SERIAL PRIMARY KEY NOT NULL,
    department_name VARCHAR(50) UNIQUE NOT NULL
);

CREATE TABLE IF NOT EXISTS doctor (
    id SERIAL PRIMARY KEY NOT NULL,
    user_id BIGINT REFERENCES user_acc (id) ON DELETE CASCADE UNIQUE NOT
NULL,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    gender VARCHAR(20) NOT NULL,
    specialization_id INT REFERENCES doctor_specialization (id) ON DELETE SET
NULL,
    department_id INT REFERENCES department (id) ON DELETE SET NULL
);

CREATE TABLE IF NOT EXISTS schedule_slot (
    id BIGSERIAL PRIMARY KEY NOT NULL,
    date_of_slot DATE NOT NULL,
    time_of_slot TIME NOT NULL,
    doctor_id INT REFERENCES doctor (id) ON DELETE CASCADE NOT NULL,
    is_taken BOOLEAN NOT NULL DEFAULT FALSE
);

CREATE TABLE IF NOT EXISTS service (
    id SERIAL PRIMARY KEY NOT NULL,
    service_name VARCHAR(50) NOT NULL,
    price DECIMAL(12,2) NOT NULL,
    doctor_id INT REFERENCES doctor (id) ON DELETE CASCADE NOT NULL
);

CREATE TABLE IF NOT EXISTS appointment (
    id BIGSERIAL PRIMARY KEY NOT NULL,
    slot_id BIGINT REFERENCES schedule_slot (id) ON DELETE CASCADE NOT NULL,
    patient_id BIGINT REFERENCES patient (id) ON DELETE CASCADE NOT NULL
);

CREATE TABLE IF NOT EXISTS appointment_service (
    id BIGSERIAL PRIMARY KEY NOT NULL,
    appointment_id BIGINT REFERENCES appointment (id) ON DELETE CASCADE NOT
NULL,
    service_id INT REFERENCES service (id) ON DELETE CASCADE NOT NULL
);

CREATE TABLE IF NOT EXISTS diagnosis (
    id SERIAL PRIMARY KEY NOT NULL,
    diagnosis_name VARCHAR(100) NOT NULL,
    diagnosis_code VARCHAR(10) NOT NULL
);

CREATE TABLE IF NOT EXISTS prescription (
    id BIGSERIAL PRIMARY KEY NOT NULL,
    note TEXT NOT NULL,

```

```

        appointment_id BIGINT REFERENCES appointment (id) ON DELETE SET NULL
    );

CREATE TABLE IF NOT EXISTS patient_diagnosis (
    id BIGSERIAL PRIMARY KEY NOT NULL,
    patient_id BIGINT REFERENCES patient (id) ON DELETE CASCADE NOT NULL,
    diagnosis_id INT REFERENCES diagnosis (id) ON DELETE CASCADE NOT NULL,
    date_of_diagnosis DATE NOT NULL DEFAULT CURRENT_DATE,
    note TEXT
);

```

Файл indexes.sql.

```

CREATE INDEX idx_role_id ON role (id);
CLUSTER role USING idx_role_id;

CREATE INDEX idx_user_acc_id ON user_acc (id);
CREATE INDEX idx_user_acc_username ON user_acc (username);
CLUSTER user_acc USING idx_user_acc_id;

CREATE INDEX idx_user_activity_id ON user_activity (id);
CREATE INDEX idx_user_activity_user_id ON user_activity (user_id);
CLUSTER user_activity USING idx_user_activity_id;

CREATE INDEX idx_patient_id ON patient (id);
CREATE INDEX idx_patient_email ON patient (email);
CLUSTER patient USING idx_patient_id;

CREATE INDEX idx_doctor_id ON doctor (id);
CLUSTER doctor USING idx_doctor_id;

CREATE INDEX idx_schedule_slot_doctor_id ON schedule_slot (doctor_id);

CREATE INDEX idx_service_id ON service (id);
CLUSTER service USING idx_service_id;

CREATE INDEX idx_appointment_id ON appointment (id);
CLUSTER appointment USING idx_appointment_id;

```

Файл triggers_and_functions.sql.

```

--user creation
CREATE OR REPLACE FUNCTION log_create_user_function()
RETURNS TRIGGER AS
$$
BEGIN
    INSERT INTO user_activity (user_id, activity_type)
    VALUES (NEW.id, 'User registered');
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS log_create_user_trigger ON user_acc;

```

```

CREATE TRIGGER log_create_user_trigger
AFTER INSERT
ON user_acc
FOR EACH ROW
EXECUTE FUNCTION log_create_user_function();

--user update
CREATE OR REPLACE FUNCTION log_update_user_function()
RETURNS TRIGGER AS
$$
BEGIN
    INSERT INTO user_activity (user_id, activity_type)
    VALUES (OLD.id, 'User updated');
    RETURN OLD;
END
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS log_update_user_trigger ON user_acc;

CREATE TRIGGER log_update_user_trigger
AFTER UPDATE
ON user_acc
FOR EACH ROW
EXECUTE FUNCTION log_update_user_function();

--take appointment
CREATE OR REPLACE FUNCTION update_slot_status_function()
RETURNS TRIGGER AS
$$
BEGIN
    UPDATE schedule_slot
    SET is_taken=TRUE WHERE id=NEW.slot_id;

    INSERT INTO user_activity (user_id, activity_type)
    VALUES (NEW.patient_id, 'User got an appointment');

    RETURN NEW;
END
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS update_slot_status_trigger ON appointment;

CREATE TRIGGER update_slot_status_trigger
AFTER INSERT
ON appointment
FOR EACH ROW
EXECUTE FUNCTION update_slot_status_function();

--cancel appointment
CREATE OR REPLACE FUNCTION cancel_slot_status_function()
RETURNS TRIGGER AS
$$
BEGIN

```

```

UPDATE schedule_slot
SET is_taken=FALSE WHERE id=OLD.slot_id;

INSERT INTO user_activity (user_id, activity_type)
VALUES (OLD.patient_id, 'User cancelled an appointment');

RETURN OLD;
END
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS cancel_slot_status_trigger ON appointment;

CREATE TRIGGER cancel_slot_status_trigger
AFTER DELETE
ON appointment
FOR EACH ROW
EXECUTE FUNCTION cancel_slot_status_function();

```

Файл queries.sql.

```

update user_acc
set password='password' where id=5;

alter table role add column new_column int;
select * from role;
alter table role drop column new_column;
select * from role;

select * from patient
where gender='Male';

select username from user_acc
where role_id=2;

select count(*) from patient
where gender='Male';

--ascending order */
select * from user_acc
order by role_id;

--descending order */
select * from user_acc
order by role_id desc;

--show full role entity instead of role_id in user_acc table */
select * from user_acc
join role on user_acc.role_id=role.id;

--show full patient info instead of patient_id in appointment table */
select * from appointment
join patient on appointment.patient_id=patient.id;

```

```
--get female patients 18+ years old */
select * from patient
where gender='Female' AND (extract(year from current_date) - extract(year
from date_of_birth)) > 17;

--the same, but shorter query */
select * from patient
where gender='Female' AND extract(year from age(date_of_birth)) > 17;

--get total appointments cost of user with id=1 */
select sum(price) from appointment_service
join service on appointment_service.service_id=service.id
join appointment on appointment_service.appointment_id=appointment.id
where patient_id=1;
```

Файл complex_queries .sql.

```
--total cost of services from catalog by doctor
SELECT
    first_name,
    last_name,
    (SELECT COALESCE(SUM(service.price), 0) FROM service WHERE
service.doctor_id = doctor.id) AS total_service_cost
FROM doctor;

--total cash by doctor
SELECT
    d.first_name,
    d.last_name,
    COALESCE(SUM(ser.price), 0) AS total_revenue
FROM doctor d
LEFT JOIN schedule_slot s ON d.id = s.doctor_id
LEFT JOIN appointment a ON s.id = a.slot_id
LEFT JOIN appointment_service aps ON a.id = aps.appointment_id
LEFT JOIN service ser ON aps.service_id = ser.id
GROUP BY d.id;

--list of patients who got appointment at least once
SELECT
    p.first_name || ' ' || p.last_name AS patient_name,
    p.date_of_birth,
    a.date_of_activity
FROM patient p
JOIN user_activity a ON p.user_id = a.user_id
WHERE a.activity_type = 'User got an appointment'
ORDER BY p.last_name, p.first_name;

--total service cost for each patient
SELECT
    p.first_name || ' ' || p.last_name AS patient_name,
    SUM(s.price) AS total_cost
FROM patient p
JOIN appointment a ON p.id = a.patient_id
```



```

JOIN appointment_service aps ON a.id = aps.appointment_id
JOIN service s ON aps.service_id = s.id
GROUP BY p.id
ORDER BY total_cost DESC;

--3 most common diagnoses
SELECT
    d.diagnosis_name,
    COUNT(pd.id) AS diagnosis_count
FROM diagnosis d
JOIN patient_diagnosis pd ON d.id = pd.diagnosis_id
GROUP BY d.id
ORDER BY diagnosis_count DESC
LIMIT 3;

--doctors who made more than 1 service
SELECT
    doctor.id,
    doctor.first_name,
    doctor.last_name,
    COUNT(appointment_service.id) AS total_services
FROM
    doctor
JOIN
    service ON doctor.id = service.doctor_id
JOIN
    appointment_service ON service.id = appointment_service.service_id
GROUP BY
    doctor.id, doctor.first_name, doctor.last_name
HAVING
    COUNT(appointment_service.id) > 1;

--average doctors' service count by specialization
SELECT
    doctor.id,
    doctor.first_name,
    doctor.last_name,
    doctor_specialization.specialization_name,
    AVG(COUNT(appointment_service.id)) OVER (PARTITION BY
doctor_specialization.id) AS avg_services_per_doctor
FROM doctor
JOIN doctor_specialization ON doctor.specialization_id =
doctor_specialization.id
LEFT JOIN service ON doctor.id = service.doctor_id
LEFT JOIN appointment_service ON service.id = appointment_service.service_id
GROUP BY doctor.id, doctor.first_name, doctor.last_name,
doctor_specialization.specialization_name, doctor_specialization.id;

--count patients by age groups
SELECT
    CASE
        WHEN extract(year from age(date_of_birth)) >= 0 AND extract(year from
age(date_of_birth)) < 18 THEN '0-17'

```

```

        WHEN extract(year from age(date_of_birth)) >= 18 AND extract(year
from age(date_of_birth)) < 30 THEN '18-29'
        WHEN extract(year from age(date_of_birth)) >= 30 AND extract(year
from age(date_of_birth)) < 50 THEN '30-49'
        WHEN extract(year from age(date_of_birth)) >= 50 THEN '50+'
        ELSE 'Unknown'
    END AS age_group,
    COUNT(*) AS patient_count
FROM patient
GROUP BY age_group
ORDER BY age_group;

```

```

--put together info about doctors' specializations and their services
EXPLAIN ANALYZE

```

```

SELECT
    d.first_name || ' ' || d.last_name AS doctor_name,
    ds.specialization_name,
    'Doctor' AS role
FROM doctor d
JOIN doctor_specialization ds ON d.specialization_id = ds.id
UNION
SELECT
    d.first_name || ' ' || d.last_name,
    s.service_name,
    'Service'
FROM doctor d
JOIN service s ON d.id = s.doctor_id;

```

```

--view for user activity
CREATE OR REPLACE VIEW patient_activity_view AS
SELECT
    p.first_name || ' ' || p.last_name AS patient_name,
    a.activity_type,
    a.date_of_activity,
    a.time_of_activity
FROM patient p
JOIN user_activity a ON p.user_id = a.user_id;

```

```

--daily report by doctor
CREATE OR REPLACE VIEW daily_report_by_doctor AS
SELECT
    d.id AS doctor_id,
    d.first_name,
    d.last_name,
    s.date_of_slot,
    COUNT(a.id) AS appointments_count
FROM doctor d
JOIN schedule_slot s ON d.id = s.doctor_id
LEFT JOIN appointment a ON s.id = a.slot_id
GROUP BY d.id, d.first_name, d.last_name, s.date_of_slot;

```

```

--patient last appointments
CREATE OR REPLACE VIEW patient_last_appointment_view AS

```

```
SELECT
    p.id AS patient_id,
    p.first_name,
    p.last_name,
    MAX(s.date_of_slot) AS last_visit_date
FROM patient p
JOIN appointment a ON p.id = a.patient_id
JOIN schedule_slot s ON a.slot_id = s.id
GROUP BY p.id, p.first_name, p.last_name;
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Конечная схема базы данных

ПРИЛОЖЕНИЕ В
(обязательное)
Ведомость курсового проекта