Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ (ИУ)
КАФЕДРА ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка базы данных для веб-приложения сети медицинских клиник

Студент ИУ7-62Б _____ Ильясов Х.М.
(группа) (подпись, дата) (И.О. Фамилия)

Руководитель курсового проекта _____ Тассов К.Л.
(подпись, дата) (И.О. Фамилия)

2024 г.

РЕФЕРАТ

Цель данной курсовой работы заключается в разработке базы данных для веб-приложения сети медицинских клиник и написании веб-приложения, которое позволяет взаимодействовать с базой данных.

В качестве системы управления базой данных использована *PostgreSQL*. Для разработки пользовательского интерфейса выбран *TypeScript React*. В качестве языка программирования выбран *Golang*.

Расчетно-пояснительная записка к курсовой работе содержит 50 страниц, 7 иллюстраций, 14 таблиц, 2 приложения, 11 источников.

Ключевые слова: медицина, клиника, врач, БД, СУБД, PostgreSQL, Golang, TypeScript, React.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Люди регулярно сталкиваются с необходимостью записаться на прием к врачу. Это может быть связано с различными обстоятельствами: от необходимости срочного медицинского вмешательства в случае травмы до планового профилактического обследования. Независимо от причины, каждый человек хочет, чтобы процесс записи прошел быстро и без лишних затруднений, а выбранный врач обладал высокой квалификацией и помог решить проблему со здоровьем.

В условиях современного мира, где технологии проникают во все сферы жизни, становится особенно важно сделать процесс записи к врачу максимально удобным и доступным. Традиционные способы, такие как телефонные звонки или личное посещение поликлиники, все чаще вызывают неудобства. Ожидание на линии, ограниченные часы работы регистратуры, необходимость выделять время на визит – все это добавляет стресса и усложняет жизнь пациентам.

Кроме того, неэффективное управление потоками пациентов приводит к перегрузке медицинского персонала и снижает качество предоставляемых услуг. В связи с этим растет потребность в разработке удобных систем, которые упростят процесс записи на прием. Цифровые платформы и специализированные приложения, предназначенные для этих целей, могут существенно изменить ситуацию в лучшую сторону. Они позволяют пациентам самостоятельно выбирать удобное время и дату визита, а также дают доступ к информации о врачах, что способствует более осознанному выбору специалиста.

Внедрение таких технологий не только экономит время, но и улучшает взаимодействие пациентов и врачей. Удобство записи и возможность заранее подготовиться к приему повышают эффективность медицинской помощи. В результате пациенты реже откладывают визит к врачу, а это, в свою очередь, способствует своевременному выявлению и лечению заболеваний.

Таким образом, внедрение современных технологий в процесс записи к врачу имеет значительные преимущества как для пациентов, так и для медицинских учреждений. Это важный шаг к созданию более доступной и качественной системы здравоохранения, которая отвечает требованиям сегодняшнего дня.

Целью курсовой работы является разработка базы данных для приложения сети медицинских клиник.

Для достижения поставленной цели необходимо решить следующие задачи:

— провести анализ существующих решений;

— формализовать задачу и определить функционал веб-приложения;

— провести анализ моделей баз данных и выбрать наиболее подходящую;

— провести анализ существующих СУБД и выбрать наиболее подходящую;

— спроектировать и разработать базу данных;

— спроектировать и разработать Web-приложение;

— исследовать зависимость времени запроса от количества записей в базе данных при наличии и отсутствии дополнительных индексов в базе.

1 Аналитическая часть

1.1 Анализ существующих решений

Рассмотрим популярные существующие решения, такие как: Поликлиника.ру, Семейный доктор, Семейная поликлиника. В таблице 1.1 представлены результаты сравнительного анализа существующих решений.

Критерии сравнительного анализа:

— а – возможность записи на прием с помощью сайта;

— б – возможность отмены записи с помощью сайта;

— в – возможность изменить дату и время записи;

Таблица 1.1 – Сравнительный анализ существующих решений

	Поликлиника.ру	**Семейный доктор**	**Семейная поликлиника**	**Предлагаемое решение**
а	+	+	-	+
б	+	-	+	+
в	-	-	-	+

Из всех представленных в таблице 1.1 аналогичных решений сеть «Поликлиника.ру» является самой полнофункциональной. Но данная сеть клиник не позволяет пользователю изменить время приема к врачу через сайт без подтверждения действия через оператора. Разрабатываемое решение устраняет недостатки этого сервиса.

1.2 Формализация задачи

В данной курсовой работе необходимо реализовать базу данных сети медицинских клиник, которая содержит в себе информацию о врачах, пациентах и их медицинских картах, а также сведения о записях пациентов к врачам. Помимо реализации базы данных необходимо разработать веб-приложение, позволяющее пользователю взаимодействовать с ней.

1.3 Формализация данных

База данных сети клиник должна включать в себя информацию о следующих категориях данных:

— пациент;

— медицинская карта пациента;

— врач;

— филиал клиники;

— кабинет филиала;

— расписание врача;

— запись пациента к врачу.

В таблице 1.2 представлены категории данных и информация о них.

Таблица 1.2 – Категории данных и информация о них

Категория	**Сведения**
Врач	ФИО, номер телефона, почта, специализация
Пациент	ФИО, номер телефона, почта, номер страховки
Медицинская карта	Хронические заболевания, аллергия, группа крови, вакцинация
Филиал	Название, номер телефона, адрес
Кабинет	Номер, этаж, филиал
Расписание врача	Врач, кабинет, дни недели
Запись пациента к врачу	Врач, пациент, дата и время

На рисунке 1.1 представлена ER-диаграмма БД в нотации Чена.

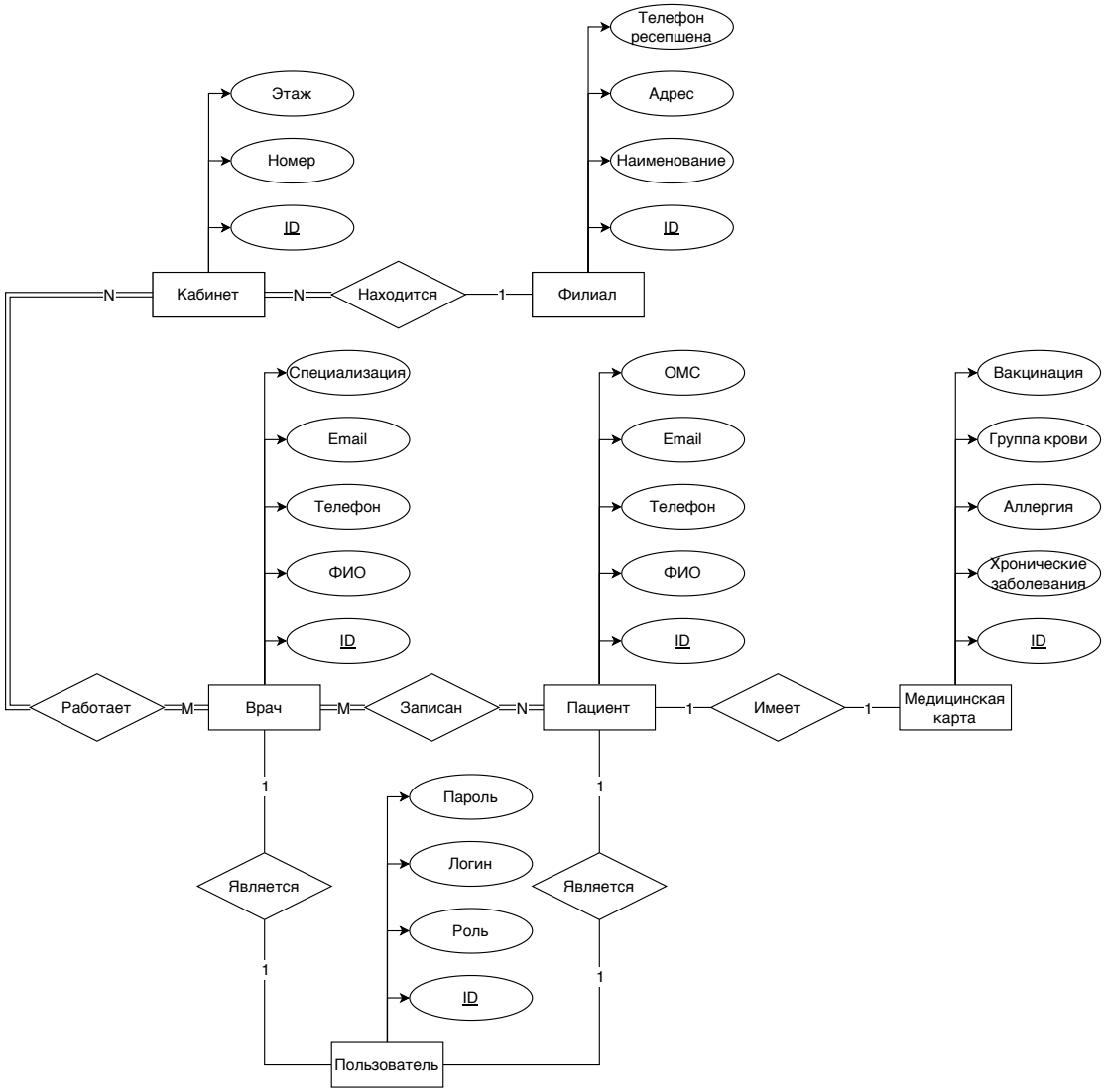

Рисунок 1.1 – ER-диаграмма в нотации Чена

1.4 Формализация пользовательских ролей

В процессе проектирования выделены следующие пользовательские роли:

— пациент;

— врач;

— администратор.

Пациент может записаться к врачу на прием, отменить или изменить свою запись, просмотреть свою медицинскую карту, а также получить информацию о своих записях к врачу. На рисунке 1.2 приведена диаграмма прецедентов для пациента.

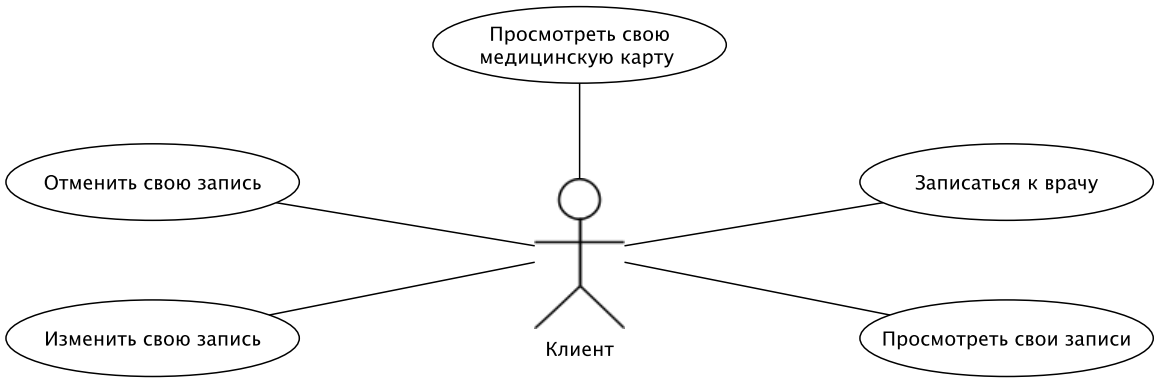

Рисунок 1.2 – Диаграмма прецедентов для пациента

Врач может получить список записанных к нему пациентов, создать медицинскую карту любого пациента, просмотреть ее, а также внести в нее изменения. На рисунке 1.3 приведена диаграмма прецедентов для врача.

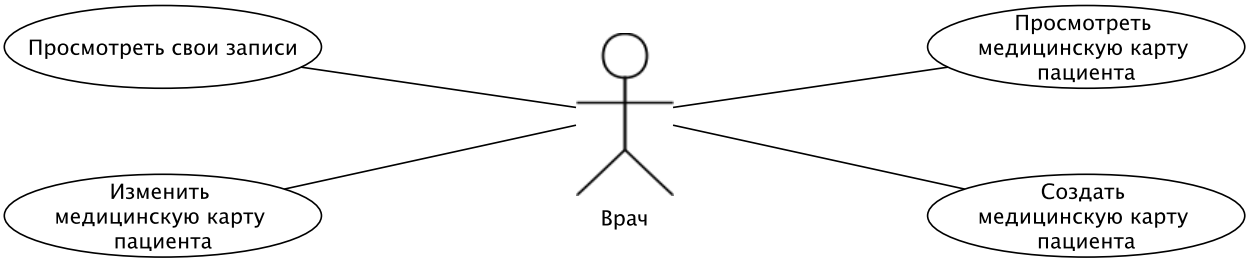

Рисунок 1.3 – Диаграмма прецедентов для врача

Роль администратора расширяет роли врача и пациента: он может выполнять все доступные действия с записями, а также получить список всех врачей и пациентов. На рисунке 1.4 приведена диаграмма прецедентов для администратора.

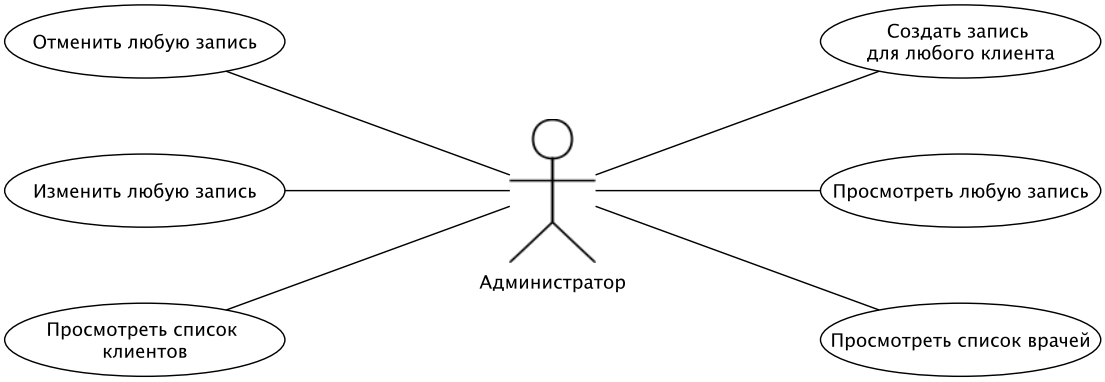

Рисунок 1.4 – Диаграмма прецедентов для администратора

1.5 Модели баз данных

База данных – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе [1]. База данных обычно управляется системой управления базами данных (СУБД). Данные вместе с СУБД, а также приложения, которые с ними связаны, называются системой баз данных, или, для краткости, просто базой данных.

Модели баз данных бывают дореляционные, реляционные и постреляционные.

1.5.1 Дореляционные модели баз данных

К дореляционным моделям БД относят иерархическую и сетевую модель, а также инвертированные списки.

Организация данных в СУБД **иерархического типа** [2] определяется в следующих терминах:

1) Атрибут (элемент данных) – наименьшая единица структуры данных. Обычно каждому элементу при описании базы данных присваивается уникальное имя. По этому имени к нему обращаются при обработке. Элемент данных также часто называют полем.

2) Запись – именованная совокупность атрибутов. Использование записей позволяет за одно обращение к базе получить некоторую логически связанную совокупность данных. Именно записи изменяются, добавляются и удаляются. Тип записи определяется составом ее атрибутов. Экземпляр записи – конкретная запись с конкретным значением элементов.

3) Групповое отношение – иерархическое отношение между записями двух типов. Родительская запись (владелец группового отношения) называется исходной записью, а дочерние записи (члены группового отношения) – подчиненными. Иерархическая база данных может хранить только такие древовидные структуры.

Корневая запись каждого дерева обязательно должна содержать ключ с уникальным значением. Ключи некорневых записей должны иметь уникальное значение только в рамках группового отношения. Каждая запись

идентифицируется полным сцепленным ключом, под которым понимается совокупность ключей всех записей от корневой по иерархическому пути.

Сетевая модель данных определяется в тех же терминах, что и иерархическая [2]. Она состоит из множества записей, которые могут быть владельцами или членами групповых отношений. Связь между записью-владельцем и записью-членом также имеет вид 1:N.

Основное различие этих моделей состоит в том, что в сетевой модели запись может быть членом более чем одного группового отношения. Согласно этой модели каждое групповое отношение именуется и проводится различие между его типом и экземпляром. Тип группового отношения задается его именем и определяет свойства, общие для всех экземпляров данного типа. Экземпляр группового отношения представляется записью-владельцем и множеством (возможно пустым) подчиненных записей. При этом имеется ограничение: экземпляр записи не может быть членом двух экземпляров групповых отношений одного типа.

Инвертированный список – это организованный в вид списка специального вида вторичный индекс, позволяющий получить всю совокупность указателей на элементы данных (записи), которым соответствует заданное значение ключа индексации [3].

Организация доступа к данным на основе инвертированных списков используется практически во всех современных реляционных СУБД, но в этих системах пользователи не имеют непосредственного доступа к инвертированным спискам (индексам).

1.5.2 Реляционные модели баз данных

Реляционная модель представляет собой совокупность данных, состоящую из набора двумерных таблиц [4]. В теории множеств таблице соответствует термин отношение, физическим представлением которого является таблица. Реляционная модель является удобной и наиболее привычной формой представления данных.

При табличной организации данных отсутствует иерархия элементов. Строки и столбцы могут быть просмотрены в любом порядке, поэтому высока гибкость выбора любого подмножества элементов в строках и столбцах.

Любая таблица в реляционной базе состоит из строк, которые называют

записями, и столбцов, которые называют полями. На пересечении строк и столбцов находятся конкретные значения данных.

1.5.3 Постреляционные модели баз данных

Постреляционная модель данных – это расширенная реляционная модель, в которой отменено требование атомарности атрибутов [5].

Она использует трехмерные структуры и позволяет хранить в полях таблицы другие таблицы. Это расширяет возможности по описанию сложных объектов реального мира.

В качестве языка запросов используется несколько расширенный SQL. Он позволяет извлекать сложные объекты из одной таблицы без операций соединения.

Вывод

В данном разделе были рассмотрены существующие решения веб-приложений медицинских клиник и предложено собственное решение, удовлетворяющее выдвинутым критериям. Также формализована задача и описаны пользовательские роли и модели баз данных. Для реализации собственного решения была выбрана реляционная модель, поскольку она представляет сущности в виде отношений. Такой способ представления данных является наиболее подходящим для решения поставленных задач.

2 Конструкторская часть

2.1 Проектирование базы данных

Ниже приведены таблицы базы данных:

— doctors – таблица врачей;

— patients – таблица пациентов;

— offices – таблица кабинетов;

— branches – таблица филиалов;

— medical_histories – таблица медицинских карт пациентов;

— users – таблица пользователей;

— appointments – таблица-связь врачей и пациентов;

— timetable – таблица-связь врачей и кабинетов.

Диаграмма базы данных представлена на рисунке 2.1.

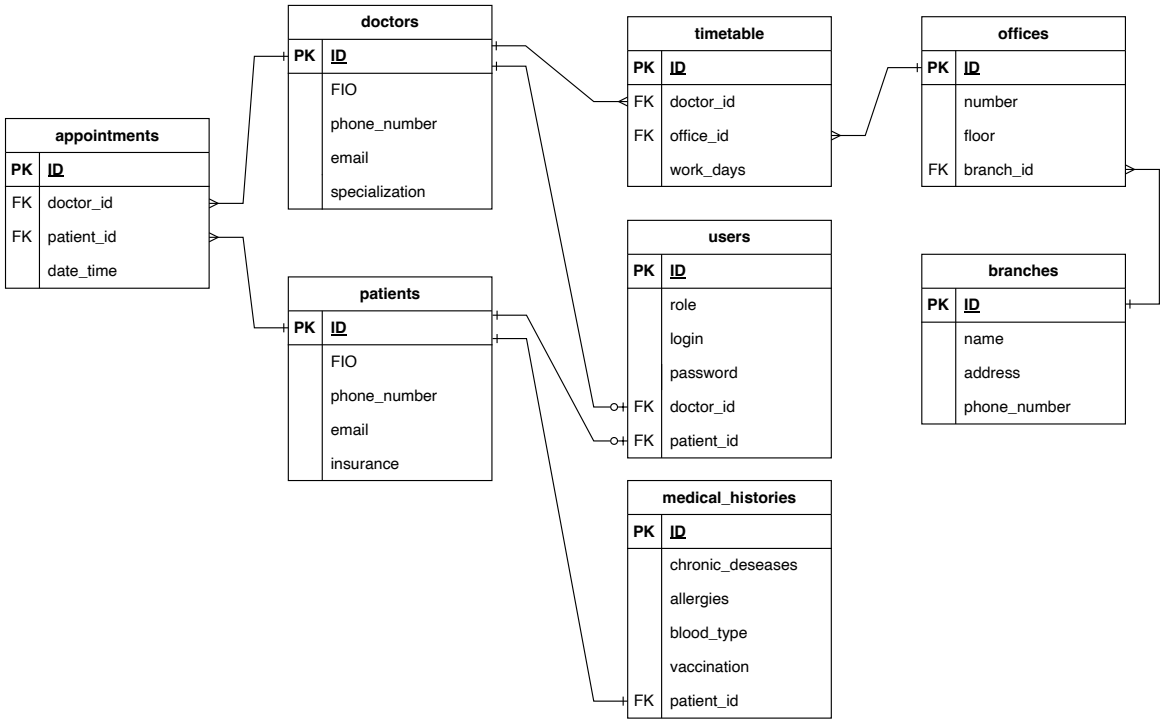

Рисунок 2.1 – Диаграмма базы данных

В таблицах 2.1–2.8 приведено описание атрибутов каждой таблицы и их ограничения целостности.

Таблица 2.1 – Описание таблицы Doctors

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор врача	Целое число	Первичный ключ, не нулевой
fio	ФИО врача	Строка	Не нулевой
phoneNumber	Номер телефона врача	Строка	Не нулевой
email	Электронная почта врача	Строка	Не нулевой
specialization	Специализация врача	Строка	Не нулевой

Таблица 2.2 – Описание таблицы Patients

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор пациента	Целое число	Первичный ключ, не нулевой
fio	ФИО пациента	Строка	Не нулевой
phoneNumber	Номер телефона пациента	Строка	Не нулевой
email	Электронная почта пациента	Строка	Не нулевой
insurance	Номер страховки пациента	Строка	Не нулевой

Таблица 2.3 – Описание таблицы Branches

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор филиала	Целое число	Первичный ключ, не нулевой
name	Название филиала	Строка	Не нулевой
phoneNumber	Номер телефона ресепшена	Строка	Не нулевой
address	Адрес филиала	Строка	Не нулевой

Таблица 2.4 – Описание таблицы Offices

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор кабинета	Целое число	Первичный ключ, не нулевой
number	Номер кабинета	Целое число	Не нулевой
floor	Этаж	Целое число	Не нулевой
branchID	Идентификатор филиала, в котором расположен кабинет	Целое число	Внешний ключ, не нулевой

Таблица 2.5 – Описание таблицы MedicalHistories

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор карты	Целое число	Первичный ключ, не нулевой
chronic_diseases	Хронические заболевания пациента	Строка	—
allergies	Аллергия	Строка	—
blood_type	Группа крови	Строка	—
vaccination	Вакцинация	Строка	—
patientID	Идентификатор пациента	Целое число	Внешний ключ, не нулевой

Таблица 2.6 – Описание таблицы Users

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор пользователя	Целое число	Первичный ключ, не нулевой
login	Логин пользователя	Строка	Не нулевой
password	Пароль пользователя	Строка	Не нулевой
role	Роль пользователя	Целое число	Не нулевой
patientID	Идентификатор пациента	Целое число	Внешний ключ
doctorID	Идентификатор врача	Целое число	Внешний ключ

Таблица 2.7 – Описание таблицы Appointments

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор записи	Целое число	Первичный ключ, не нулевой
doctorID	Идентификатор врача	Целое число	Внешний ключ, не нулевой
patientID	Идентификатор пациента	Целое число	Внешний ключ, не нулевой
datetime	Дата и время записи	timestamp	Не нулевой

Таблица 2.8 – Описание таблицы Timetable

Атрибут	**Назначение**	**Тип данных**	**Ограничение**
id	Идентификатор записи	Целое число	Первичный ключ, не нулевой
doctorID	Идентификатор врача	Целое число	Внешний ключ, не нулевой
officeID	Идентификатор кабинета	Целое число	Внешний ключ, не нулевой
workDays	Рабочий день	Целое число	Не нулевой

2.2 Хранимая функция базы данных

Хранимая функция – это набор SQL-выражений, которые выполняют некоторую операцию и возвращают значение.

Разработанная функция *add_appointment*() предназначена для создания новой записи приема пациента к врачу в определенное время. Она принимает 3 параметра: идентификатор врача, идентификатор пациента и дату и время записи на прием. В теле функции выполняется проверка существования врача и пациента с такими идентификаторами. В случае успеха создается новая запись в таблице Appointments с заданными значениями полей.

Схема алгоритма работы хранимой функции приведена на рисунке 2.2.

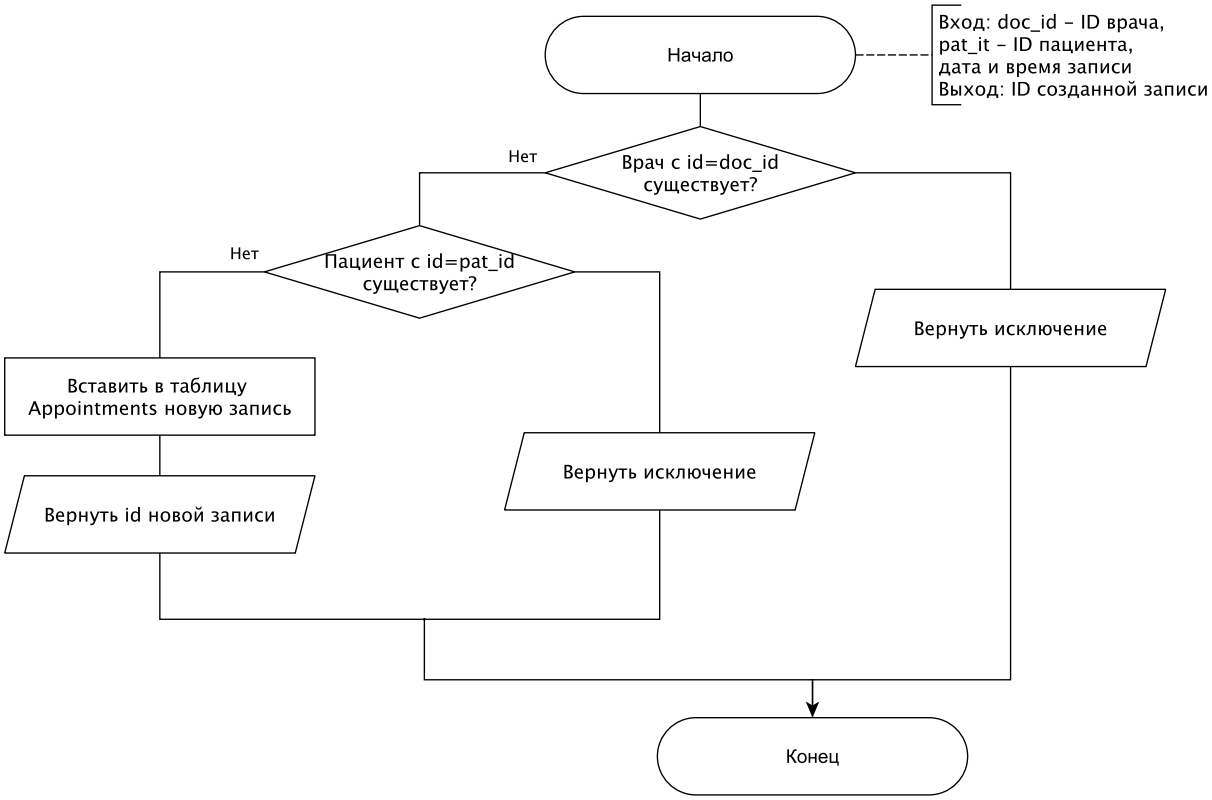

Рисунок 2.2 – Схема алгоритма работы хранимой функции создания записи к врачу

2.3 Роли уровня базы данных

Для каждой из ролей сервиса, описанных ранее, необходимо создать роль уровня базы данных.

— Врач может просмотреть информацию о своих записях, а также создать медицинскую карту пациента, просмотреть информацию о ней и вносить изменения в медицинскую карту определенного пациента. Следовательно, врач должен иметь возможность выполнять операции SELECT в таблице Appointments и операции SELECT, UPDATE и INSERT в таблице MedicalHistories, а также врач может выполнять операцию SELECT в таблице Patients, поскольку ему нужно знать данные пациента, которого нужно осмотреть.

— Пациент может записаться на прием к определенному врачу в определенное время, просмотреть информацию о всех своих записях, а также изменить информацию о них или отменить их. Следовательно, пациент должен иметь возможность выполнять операции SELECT, INSERT, UPDATE и DELETE в таблице Appointments. Поскольку пациент должен знать имена врачей, к которым он записывается, а также дни их работы, то у него должна быть возможность получить информацию из таблиц Doctors и Timetable с помощью операции SELECT.

— Администратор может создать, удалить или изменить запись, просмотреть информацию об определенной записи по идентификатору, получить список всех врачей, список всех пациентов, внести изменения в любую таблицу или добавить/удалить информацию в таблицах. Следовательно, администратор должен иметь возможность выполнять операции SELECT, INSERT, UPDATE и DELETE во всех существующих таблицах без ограничений.

Вывод

В данном разделе была спроектирована база данных, описаны таблицы и хранимая функция, а также выделены роли уровня базы данных.

3 Технологическая часть

3.1 Средства реализации

В данной работе для написания Web-приложения был выбран язык программирования *Golang* [6] ввиду следующих причин:

— компиляция в машинный код: Go компилируется непосредственно в машинный код, что обеспечивает высокую скорость выполнения приложений;

— эффективное использование памяти: Go оптимизирует использование памяти, что позволяет веб-приложениям работать более эффективно [7];

— обнаружение ошибок на этапе компиляции: статическая типизация позволяет выявлять ошибки типов до выполнения программы, что повышает надежность и безопасность кода;

— явное управление типами: явное указание типов данных улучшает читаемость и понимание кода;

— широкий набор стандартных библиотек: Go поставляется с обширной стандартной библиотекой, включающей поддержку работы с HTTP, JSON, базами данных, шаблонами;

— инструменты для тестирования и профилирования: встроенные инструменты для тестирования, профилирования и анализа кода помогают улучшить качество разработки;

— кросс-компиляция: Go легко компилируется для различных платформ, что облегчает создание приложений для различных операционных систем.

В качестве среды разработки был выбран *JetBrains Goland* [8] по следующим причинам:

— интеграция с lint-инструментами: поддержка GoLint и других инструментов для статического анализа, которые могут быть интегрированы для улучшения качества кода;

— поддержка Git: полная интеграция с системой контроля версий, включая инструменты для управления ветками, слияний и просмотра истории коммитов;

— мощный отладчик: GoLand предоставляет инструменты для пошаговой отладки, установки точек останова и анализа состояния программы во время выполнения;

— интеграция с Docker: поддержка создания, управления и отладки Docker-контейнеров прямо из IDE;

— мульти-языковая поддержка: GoLand поддерживает HTML, CSS и TypeScript, что удобно для разработки веб-приложений;

— поддержка фреймворков и библиотек: интеграция с фреймворками и библиотеками для веб-разработки, такими как React, Angular и Vue.js.

При выборе средства для разработки пользовательского интерфейса был выбран фреймворк *TypeScript React* [9] по следующим причинам:

— статическая типизация: TypeScript позволяет определить типы данных, что помогает предотвратить ошибки на этапе компиляции;

— раннее обнаружение ошибок: TypeScript позволяет находить ошибки типов и синтаксиса на этапе компиляции, до выполнения кода, что уменьшает количество ошибок при выполнении;

— подробная документация: React достаточно полно описан в официальной документации, что упрощает его изучение.

3.2 Выбор СУБД

Выбор СУБД проводился среди MSSQL, Oracle, PostgreSQL и MySQL.

В таблице 3.1 приведен сравнительный анализ рассматриваемых СУБД, который проводился по следующим критериям:

1) бесплатное распространение СУБД;

2) возможность создания ролевой модели;

3) наличие опыта работы с СУБД.

Таблица 3.1 – Сравнительный анализ СУБД

Критерий	MSSQL	Oracle	PostgreSQL	MySQL
1	-	-	+	+
2	+	+	+	+
3	-	-	+	-

В результате сравнительного анализа было принято решение использовать СУБД PostgreSQL [10], поскольку она удовлетворяет всем обозначенным критериям.

3.3 Реализация хранимой функции, таблиц и ролей базы данных

В листинге 5.1 приложения Б приведена реализация создания хранимой функции *add_appointment*() для добавления записи в таблицу Appointments.

В листинге 5.2 приложения Б приведена реализация создания таблиц базы данных.

В листинге 5.3 приложения Б приведена реализация создания ролей уровня базы данных.

3.4 Тестирование хранимой функции

Для тестирования хранимой функции *add_appointment*(), реализация которой представлена в листинге 5.1, было принято решение использовать язык программирования *Python* и библиотеку *psycopg*2 [11], которая предоставляет интерфейс для работы с базой данных.

Написанный скрипт выполняет следующие действия:

— создает подключение к базе данных с тестируемой функцией;

— вызывает хранимую функцию, которая выполняет добавление новой записи в таблицу Appointments;

— сравнивает содержимое таблицы Appointments до и после вызова функции.

В листинге 5.4 приложения Б приведена реализация описанного выше скрипта для тестирования хранимой функции.

3.5 Автоматизация развертывания

Для автоматизации развертывания приложения и управления им использованы инструменты Docker и Docker Compose.

Docker – это платформа для автоматизации развертывания приложений в контейнерах, обеспечивающая изоляцию приложения и его зависимостей от системы. Контейнеры Docker позволяют гарантировать, что приложение будет работать одинаково на всех окружениях – от локальной разработки до тестовых и производственных серверов.

Docker Compose – это инструмент для определения и управления многоконтейнерными Docker-приложениями. С помощью Docker Compose можно описать всю инфраструктуру приложения, включая базы данных, в одном файле и запускать их одной командой.

Для удобства развертывания и управления зависимостями в рамках проекта был создан файл docker-compose.yml, который включает описание сервисов базы данных и основного приложения.

Листинг описанного выше файла приведен в листинге 5.5.

4 Исследовательская часть

4.1 Описание исследования

Индекс – это объект базы данных, предназначенный для сокращения времени выполнения запроса и обеспечивающий быстрый доступ к данным. Индексы являются указателями, которые позволяют эффективно находить и извлекать записи, не прибегая к полному сканированию таблицы. Они играют ключевую роль в оптимизации работы баз данных при обработке больших объемов информации.

Было принято решение провести исследование зависимости времени запроса от количества записей в базе данных при наличии и отсутствии дополнительных индексов. Для исследования влияния индекса на время выполнения запроса поиска создан индекс на столбцах ($doctor\_id$, $patient\_id$) таблицы Appointments.

Исследовался запрос, приведенный в листинге 4.1

Листинг 4.1 – Исследованный запрос

```
SELECT * FROM appointments
WHERE doctorid = 200 and patientid = 200;
```

Время выполнения запроса в зависимости от количества записей в таблице без дополнительных индексов приведено в таблице 4.1.

Таблица 4.1 – Замер времени при отсутствии дополнительных индексов

Количество записей	**Время запроса, мкс**
10	686.027
50	725.549
100	769.638
250	897.928
500	1013.807
1000	1206.138
2500	1391.622
5000	1506.001
7500	1621.913
10000	1714.385
25000	1858.714
50000	1992.788
100000	2101.576

Время выполнения запроса в зависимости от количества записей в таблице с использованием дополнительных индексов приведено в таблице 4.2.

Таблица 4.2 – Замер времени при наличии дополнительных индексов

Количество записей	**Время запроса, мкс**
10	584.876
50	637.154
100	659.758
250	679.879
500	691.512
1000	717.845
2500	840.665
5000	992.659
7500	1089.317
10000	1136.622
25000	1244.534
50000	1346.117
100000	1528.174

На рисунке 4.1 представлен график, иллюстрирующий зависимость времени выполнения запроса от количества записей в таблице при наличии и отсутствии дополнительных индексов.

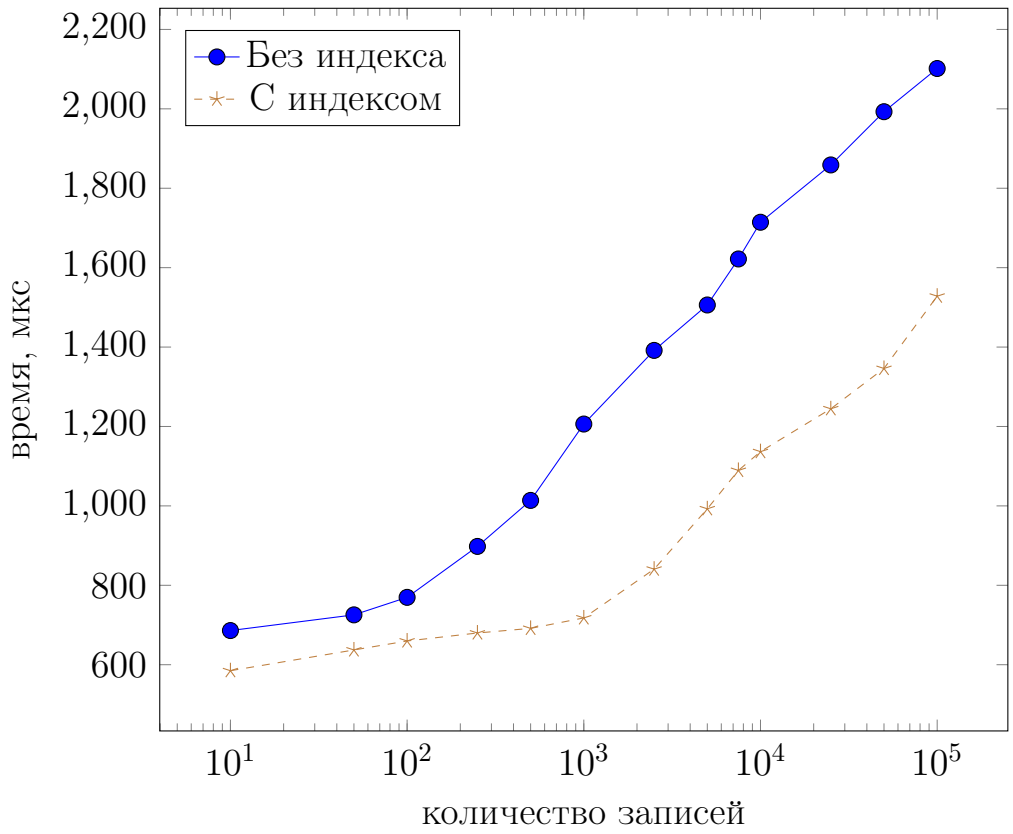

Рисунок 4.1 – Сравнение времени выполнения запроса поиска при отсутствии и наличии доп. индекса

В результате анализа данных, представленных в таблицах 4.1 и 4.2, сделан следующий вывод: скорость выполнения запроса поиска в таблице с дополнительным индексом выше, чем скорость выполнения этого запроса в таблице без индекса.

В таблице 4.3 приведен результат сравнения времени выполнения запроса в таблицах с дополнительным индексом и без него при 1000, 10000, 25000, 50000 и 100000 записей:

Таблица 4.3 – Результат сравнения при 1000, 10000, 25000, 50000 и 100000 записей в таблицах

Количество записей	$t_{\text{без индекса}}/t_{\text{с индексом}}$
1000	1.68
10000	1.51
25000	1.49
50000	1.48
100000	1.38

Таким образом, время выполнения запроса в таблице без дополнительного индекса, содержащей 100000 записей, в 3.06 раз выше, чем время выполнения запроса в такой таблице, содержащей 10 записей. В таблице с дополнительным индексом при том же количестве записей значение данного коэффициента равно 2.61.

Вывод

Из результатов проведенного исследования следует вывод: выполнение запроса в таблицу без дополнительных индексов требует в среднем на 42 % больше времени, чем выполнение того же запроса в таблице с индексом.

ЗАКЛЮЧЕНИЕ

В рамках курсовой работы было разработано программное обеспечение для взаимодействия с базой данных сети медицинских клиник, позволяющее получить доступ к информации о записях, врачах и пациентах и изменить ее.

При увеличении количества записей выполнение запроса в таблице с дополнительным индексом эффективнее примерно на 17.2 %.

В ходе выполнения курсовой работы были решены все поставленные задачи:

— проведен анализ существующих решений;

— формализована задача и определен функционал веб-приложения;

— проведен анализ моделей баз данных и выбрана наиболее подходящая;

— проведен анализ существующих СУБД и выбрана наиболее подходящая;

— спроектирована и разработана база данных;

— спроектировано и разработано Web-приложение;

— проведено исследование зависимости времени запроса от количества записей в базе данных при наличии и отсутствии дополнительных индексов в базе.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое база данных? [Электронный ресурс]. Режим доступа: https://www.oracle.com/cis/database/what-is-database/ (дата обращения: 26.05.2024).

2. Модели баз данных [Электронный ресурс]. Режим доступа: https://koi.tspu.ru/inf_syst_shem.htm (дата обращения: 26.05.2024).

3. Ранние СУБД [Электронный ресурс]. Режим доступа: http://algmet.simulacrum.me/subd/lect3.htm (дата обращения: 26.05.2024).

4. Реляционная модель данных [Электронный ресурс]. Режим доступа: https://bseu.by/it/tohod/lekcii2_3.htm (дата обращения: 26.05.2024).

5. Постреляционные СУБД [Электронный ресурс]. Режим доступа: https://dit.isuct.ru/IVT/BOOKS/DBMS/DBMS14/ch_6_2.html (дата обращения: 26.05.2024).

6. Golang Documentation [Электронный ресурс]. Режим доступа: https://go.dev/doc/ (дата обращения: 26.05.2024).

7. Язык программирования Golang – что это? [Электронный ресурс]. Режим доступа: https://lemon.school/ru/blog/mova-programuvannya-golang-shho-tse (дата обращения: 26.05.2024).

8. JetBrains Goland [Электронный ресурс]. Режим доступа: https://www.jetbrains.com/ru-ru/go/ (дата обращения: 26.05.2024).

9. TypeScript: Documentation – React [Электронный ресурс]. Режим доступа: https://www.typescriptlang.org/docs/handbook/react.html (дата обращения: 26.05.2024).

10. PostgreSQL Documentation [Электронный ресурс]. Режим доступа: https://www.postgresql.org/docs/ (дата обращения: 26.05.2024).

11. psycopg2 [Электронный ресурс]. Режим доступа: https://www.psycopg.org/docs/ (дата обращения: 26.05.2024).

ПРИЛОЖЕНИЕ А

Презентация содержит 12 страниц.

ПРИЛОЖЕНИЕ Б

Листинг 5.1 – Создание хранимой функции

```
CREATE OR REPLACE FUNCTION add_appointment(
    p_doctorID INT,
    p_patientID INT,
    p_datetime TIMESTAMP
)
    RETURNS INT
    LANGUAGE plpgsql
AS $$
DECLARE
    v_doctorExists BOOLEAN;
    v_patientExists BOOLEAN;
    v_appointmentID INT;
BEGIN
    SELECT EXISTS(SELECT 1 FROM doctors WHERE id = p_doctorID) INTO
       v_doctorExists;
    IF NOT v_doctorExists THEN
        RAISE EXCEPTION 'Doctor with ID % does not exist', p_doctorID;
    END IF;

    SELECT EXISTS(SELECT 1 FROM patients WHERE id = p_patientID) INTO
       v_patientExists;
    IF NOT v_patientExists THEN
        RAISE EXCEPTION 'Patient with ID % does not exist', p_patientID;
    END IF;

    INSERT INTO appointments (doctorID, patientID, datetime)
    VALUES (p_doctorID, p_patientID, p_datetime)
    RETURNING id INTO v_appointmentID;

    RETURN v_appointmentID;
END;
$$;
```

Листинг 5.2 – Создание таблиц (начало)

```
CREATE TABLE IF NOT EXISTS dicdoc_service.public.doctors (
    id SERIAL PRIMARY KEY,
    fio TEXT NOT NULL,
    phoneNumber TEXT NOT NULL,
    email TEXT NOT NULL,
    specialization TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS dicdoc_service.public.patients (
    id SERIAL PRIMARY KEY,
    fio TEXT NOT NULL,
    phoneNumber TEXT NOT NULL,
    email TEXT NOT NULL,
    insurance TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS dicdoc_service.public.branches (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    address TEXT NOT NULL,
    phoneNumber TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS dicdoc_service.public.offices (
    id SERIAL PRIMARY KEY,
    number INT NOT NULL,
    floor INT NOT NULL,
    branchID INT NOT NULL,
    FOREIGN KEY (branchID) REFERENCES branches(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS dicdoc_service.public.timetable (
    id SERIAL PRIMARY KEY,
    doctorID INT NOT NULL,
    officeID INT NOT NULL,
    workDays INT NOT NULL,
    FOREIGN KEY (doctorID) REFERENCES doctors(id) ON DELETE CASCADE,
    FOREIGN KEY (officeID) REFERENCES offices(id) ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS dicdoc_service.public.appointments (
    id SERIAL PRIMARY KEY,
    doctorID INT NOT NULL,
    patientID INT NOT NULL,
    datetime TIMESTAMP NOT NULL,
    FOREIGN KEY (doctorID) REFERENCES doctors (id) ON DELETE CASCADE,
    FOREIGN KEY (patientID) REFERENCES patients (id) ON DELETE CASCADE
);
```

Листинг 5.2 – Создание таблиц (конец)

```
CREATE TABLE IF NOT EXISTS dicdoc_service.public.users (
    id SERIAL PRIMARY KEY,
    login TEXT NOT NULL,
    password TEXT NOT NULL,
    role INT NOT NULL,
    patientID INT,
    doctorID INT,
    FOREIGN KEY (patientID) REFERENCES patients(id) ON DELETE CASCADE,
    FOREIGN KEY (doctorID) REFERENCES doctors(id) ON DELETE CASCADE
);

CREATE TABLE IF NOT EXISTS dicdoc_service.public.medical_histories (
    id SERIAL PRIMARY KEY,
    chronic_diseases text,
    allergies text,
    blood_type text,
    vaccination text,
    patientID INT NOT NULL,
    FOREIGN KEY (patientID) REFERENCES patients(id) ON DELETE CASCADE
);
```

Листинг 5.3 – Создание ролей (начало)

```
CREATE ROLE admin WITH
    SUPERUSER
    NOINHERIT
    NOREPLICATION
    NOBYPASSRLS
    LOGIN
    PASSWORD 'admin';

GRANT ALL PRIVILEGES
    ON ALL TABLES IN SCHEMA public
    TO admin;

CREATE ROLE doctor WITH
    NOSUPERUSER
    NOINHERIT
    NOCREATEROLE
    NOCREATEDB
    NOBYPASSRLS
    NOREPLICATION
    LOGIN
    PASSWORD 'doc';

GRANT SELECT
    ON TABLE public.appointments TO doctor;
```

Листинг 5.3 – Создание ролей (конец)

```
GRANT SELECT, UPDATE, INSERT
    ON TABLE public.medical_histories TO doctor;

GRANT SELECT
    ON TABLE public.patients TO doctor;

CREATE ROLE patient WITH
    NOSUPERUSER
    NOINHERIT
    NOCREATEROLE
    NOCREATEDB
    NOBYPASSRLS
    NOREPLICATION
    LOGIN
    PASSWORD 'patient';

GRANT SELECT, INSERT, UPDATE, DELETE
    ON TABLE public.appointments TO patient;

GRANT SELECT
    ON TABLE public.doctors TO patient;

GRANT SELECT
    ON TABLE public.timetable TO patient;
```

Листинг 5.4 – Скрипт для тестирования хранимой функции

```
import psycopg2
from psycopg2 import sql

def get_table_data(cursor, table_name):
    cursor.execute(sql.SQL("SELECT * FROM {}")
                   .format(sql.Identifier(table_name)))
    return cursor.fetchall()

def main():
    conn = psycopg2.connect(
        dbname="dicdoc_service",
        user="postgres",
        password="postgres",
        host="localhost",
        port="5432"
    )
    cursor = conn.cursor()

    table_name = 'appointments'

    initial_data = get_table_data(cursor, table_name)

    doc_id = pat_id = 100
    datetime = "2024-06-11 19:00:00.000000"

    cursor.execute("SELECT * FROM add_appointment(%s, %s, %s)", (doc_id,
       pat_id, datetime,))
    conn.commit()

    final_data = get_table_data(cursor, table_name)

    initial_set = set(initial_data)
    final_set = set(final_data)

    print(sorted(list(initial_data)), sorted(list(final_data)), sep="\n\n")

    added_record = final_set - initial_set

    cursor.close()
    conn.close()

    print("\nAdded: ", added_record)

if __name__ == "__main__":
    main()
```

Листинг 5.5 – Содержимое файла docker-compose.yml

```
services:
  postgres:
    image: postgres:latest
    container_name: postgres
    restart: on-failure
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: "postgres"
      POSTGRES_DB: postgres
    volumes:
      - ./scripts/1-init.sql:/docker-entrypoint-initdb.d/1-init.sql
      - db-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  dicdoc-service:
    container_name: dicdoc-service
    build:
      context: ./
    ports:
      - "8080:8080"
    depends_on:
      - postgres
    links:
      - postgres
    restart: on-failure
    env_file:
      - configs/cfg.env

volumes:
  db-data:
```