



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»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

**”Разработка веб сайта для управления волонтерской
организацией”**

Студент Назирова И.В.

Группа ИУ7-65Б

Научный руководитель Кивва К.А.

Подпись руководителя _____

Москва — 2023 г.

РЕФЕРАТ

Отчёт содержит 69 стр., 26 рис., 3 табл., 8 источн.

Курсовая работа представляет собой разработку базы данных для управления волонтерской организацией, а также приложения, предоставляющего интерфейс для доступа к базе данных.

В качестве системы управления базами данных используется PostgreSQL, которая подключается к приложению, реализованному на языке программирования Java.

Ключевые слова: база данных, PostgreSQL, Java, Typescript, React, Redux, Linux, система управления базами данных

СОДЕРЖАНИЕ

РЕФЕРАТ	3
Введение	6
1 Аналитическая часть	8
1.1 Анализ предметной области	8
1.2 Формализация данных	10
1.3 Формализация ролей	12
1.4 Определение понятия СУБД	15
1.4.1 Основные функции СУБД	15
1.5 Классификация СУБД по модели данных	16
1.5.1 Иерархическая база данных	16
1.5.2 Графовые базы данных	18
1.5.3 Реляционные базы данных	20
1.6 Вывод	22
2 Конструкторская часть	23
2.1 Проектирование базы данных	23
2.2 Требования к программе	27
2.3 Проектирование приложения	28
2.4 Проектирование хранимых в БД функции/процедур	32
2.5 Проектирование хранимых в БД триггеров	33
2.6 Вывод	34
3 Технологическая часть	35
3.1 Выбор и обоснование используемых технологий	35
3.2 Создание объектов БД	36
3.2.1 Создание таблиц	36
3.2.2 Создание связей	41
3.2.3 Создание ролей и выделение им прав	43
3.2.4 Создание функций	47
3.2.5 Создание триггеров	49
3.3 Графический интерфейс	50
4 Исследовательский раздел	57
4.1 Цель эксперимента	57
4.2 Технические характеристики	57

4.3	Описание эксперимента	57
4.4	Результаты эксперимента	63
4.5	Вывод	67
	Заключение	68
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	69

Введение

Волонтерство становится все более популярным среди студентов и молодежи. Согласно исследованию [1], проведенному в 2021 году, более 60% молодежи выражают желание участвовать в волонтерских проектах и организациях. Данный проект предназначен для организации волонтерских кружков, что может привлечь еще больше студентов к участию в волонтерстве.

По данным опроса, проведенного среди студентов [1], более 80% студентов считают, что наличие приложения для организации волонтерских кружков упрощает процесс поиска кружков и записи на них. Это подтверждает актуальность проекта и необходимость его создания.

Также стоит отметить, что волонтерство может быть важным фактором при поступлении в вузы и при трудоустройстве. Согласно исследованию [1], проведенному в 2020 году, более 70% работодателей принимают во внимание опыт волонтерства при рассмотрении кандидатов на должности. Поэтому удобное приложение для организации волонтерских кружков может помочь студентам получить ценный опыт и улучшить свои шансы найти будущую работу.

Анализ предметной области проекта показывает, что существует необходимость в удобном приложении для кураторов, студентов и волонтеров волонтерской организации, которое бы позволяло эффективно организовывать кружки и управлять ими.

Цель данного проекта - разработка базы данных и ПО, которое позволит куратору организовывать кружки. В проекте участвуют три группы пользователей: студенты, кураторы и волонтеры. Для каждой группы пользователей предусмотрены свои функции в приложении. Для студентов доступен просмотр списка кружков, их расписания, а также возможность записаться в понравившийся кружок. Для кураторов предусмотрены функции создания и изменения кружков, добавление и удаление волонтеров, а также установление количества мест в кружке. Для волонтеров доступен просмотр списка кружков, их расписания, а также список участников.

Данное приложение будет полезно для волонтерской организации, которая занимается организацией кружков для студентов. Оно позволит упростить процесс организации и контроля за кружками, а также обеспечить более удобный доступ к информации для всех участников процесса.

Для реализации проекта необходимо:

- провести анализ предметной области и готовых решений в области баз данных и систем управления базами данных;
- описать все необходимые сущности и связи между ними;
- спроектировать базу данных;
- выбрать подходящую систему управления базами данных и разработать интерфейс для доступа к базе данных;
- реализовать программу. Программа должна обеспечивать безопасность данных, а также быть легко масштабируемой и расширяемой для дальнейшей разработки;
- провести исследование устойчивости реализованной системы к высоким нагрузкам.

1 Аналитическая часть

В данном разделе проведён анализ предметной области, проведена классификация баз данных (БД) и систем управления базами данных (СУБД).

1.1 Анализ предметной области

Исследования различных типов волонтерских организаций и их структуры. Волонтерские организации могут быть разнообразными, например, общественными, благотворительными, экологическими и т.д [1]. Каждая организация имеет свои цели, миссию и специфику деятельности. Структура волонтерской организации может включать [1] руководство (например, председатель, исполнительный директор), различные отделы или комитеты (например, финансовый, маркетинговый, программный), а также волонтеров, которые активно участвуют в реализации проектов организации.

Анализ существующих решений в области автоматизации организации кружков и работы с волонтерами (Таблица 1). Существуют различные веб-платформы, приложения и информационные системы, предназначенные для управления волонтерскими программами и координации деятельности кураторов, студентов и волонтеров. Эти решения обычно предоставляют функционал, который позволяет управлять списком кружков, расписанием, регистрацией участников, взаимодействием между кураторами, студентами и волонтерами, а также предоставляют отчетность о проделанной работе [1].

Сущность Куратор является ключевой для данной предметной области, поскольку именно куратор организует кружки и контролирует процесс их проведения. Сущность Волонтер также играет важную роль в предметной области. Волонтеры - это люди, готовые добровольно участвовать в различных кружках и помогать кураторам в их проведении. Сущность Студент является важной сущностью в предметной области управления кружками и волонтерской деятельностью. Студенты являются участниками кружков и активно взаимодействуют с кураторами и другими участниками. Кружок же представляет собой сущность, описывающую отдельный кружок, на который может записаться студент.

Функционал приложения должен удовлетворять потребности всех участников организации кружков. Для студентов приложение должно предоставлять возможность просмотра списка кружков, их расписания, а также записаться в понравившийся кружок. Для кураторов приложение должно предоставлять возмож-

ность просмотра списка студентов, создание/изменение кружков, добавление/удаление волонтеров, а также возможность устанавливать количество мест в кружке. Для волонтеров приложение должно предоставлять возможность просмотра списка кружков, их расписания, а также список участников.

Таблица 1 – Сравнения существующих аналогов

Решение	Функциональность	Преимущества	Недостатки
Volunteer Management System	Управление волонтерскими программами Регистрация и учёт волонтеров Расписание и отчётность	Разные уровни доступа Функции управления списком кружков и участников Веб-платформа	Может быть ограничен функционалом, специфичным для волонтерской организации
Volunteer Hub	Управление волонтерами Управление кружками и расписанием Коммуникация и отчётность	Интегрированная система Разные уровни доступа Генерация отчётов Интерфейс для всех ролей	Может быть сложным в настройке и интеграции с другими системам
Volunteer Connect	Поиск и регистрация волонтеров Управление кружками и расписанием Отчётность	Удобный интерфейс Функции поиска и регистрации волонтеров Возможность создания своего кружка	Может быть ограничен функционалом, специфичным для управления кружками

1.2 Формализация данных

База данных должна содержать информацию о следующих объектах (Таблица 2).

- а) пользователях
- б) ролях
- в) клубах (кружки)
- г) событиях
- д) новостях
- е) финансов (финансы волонтерской организации)
- ж) материалах

Модель предметной области в классической нотации Питера Чена представлена на рисунке 1.

Таблица 2 – Формализация данных

Сущность	Сведения
Пользователь	идентификатор, логин, пароль, электронная почта
Информация о пользователе	идентификатор, идентификатор пользователя, имя, фамилию, номер телефона, пол, возраст, образование, опыт работы, дата регистрации
Роль	идентификатор, наименование роли
События	идентификатор, название, дата проведение, аудитория, количество мест, описание
Клубы	идентификатор, название, уровень сложности, продолжительность, время начала занятия, время конца занятия, день недели, количество мест, требования, описание
Финансы	идентификатор, дата создание, описание, сумма, тип (доход или расход), программа, участники, волонтера
Материалы	идентификатор, название, описание, формат, автор, дата создание
Новости	идентификатор, заголовок, содержание, дата создание

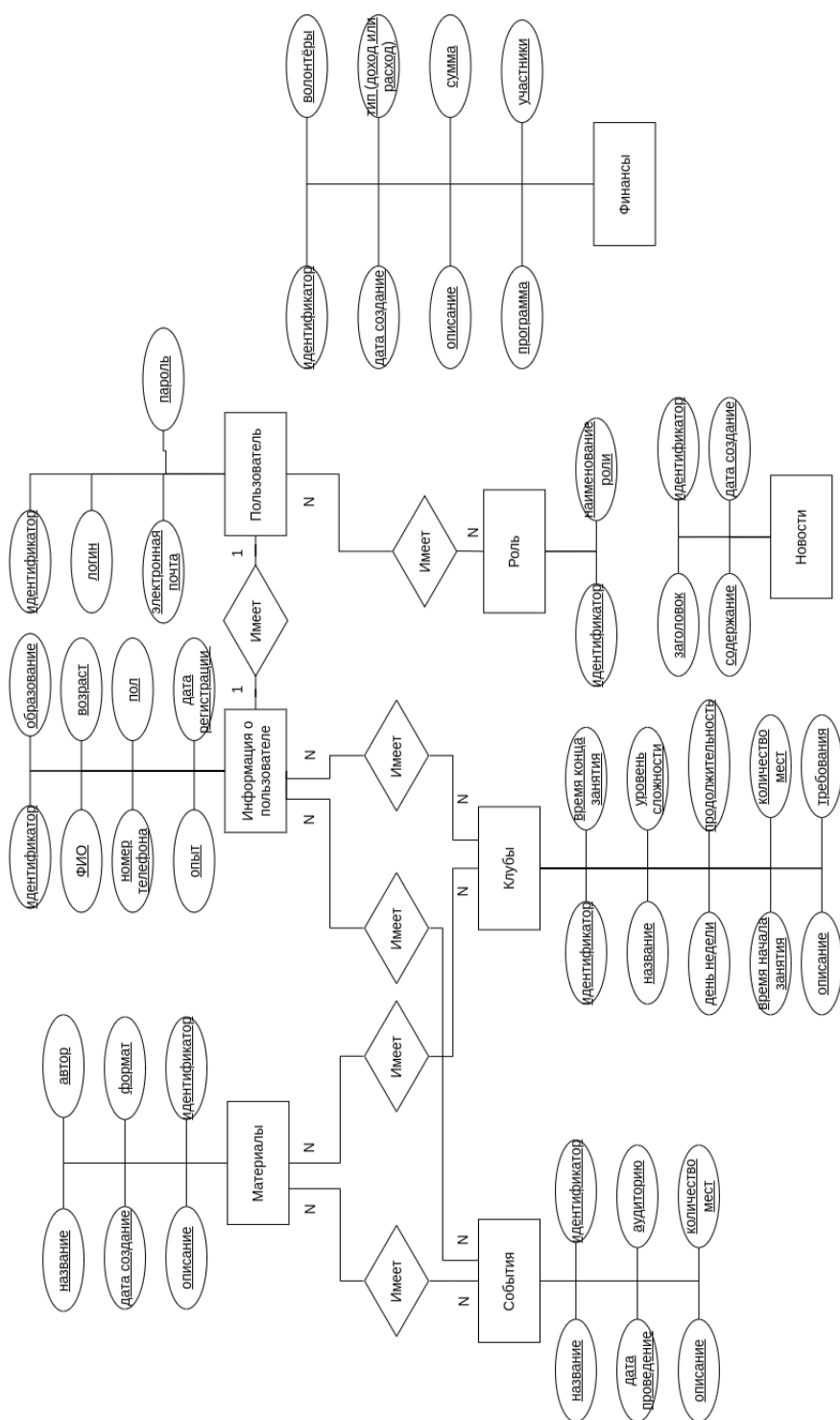


Рисунок 1 – Модель предметной области в классической нотации Питера Чена

1.3 Формализация ролей

Для реализации личного кабинета, необходима система авторизации, которая позволит различать авторизованных и неавторизованных пользователей (Таблица 3).

В таблице 3 и на рисунках (2 - 5) представлены возможности пользователей и описание системы на концептуальном уровне в виде диаграммы прецедентов.

Таблица 3 – Формализация ролей

Тип пользователя	Функционал
Неавторизованный	<p>Зайти на страницу регистрации, зарегистрироваться или перейти на страницу авторизации.</p> <p>Зайти на страницу авторизации, авторизоваться или перейти на страницу регистрации.</p> <p>Посмотреть расписание.</p> <p>Авторизоваться, сделать что-то из предыдущих действий или ни одно из них, выйти из аккаунта.</p>
Авторизованный (куратор)	<p>Зайти на страницу своего профиля.</p> <p>Зайти на страницу добавить / изменить кружок и количество мест в этом кружке.</p> <p>Зайти на страницу посмотреть список студентов и список студентов конкретного кружка.</p>
Авторизованный (волонтер)	<p>Зайти на страницу своего профиля.</p> <p>Зайти на страницу посмотреть список студентов и список студентов конкретного кружка.</p>
Авторизованный (студент)	<p>Зайти на страницу своего профиля.</p> <p>Зайти на страницу своего профиля, записаться / покинуть кружок.</p> <p>Зайти на страницу посмотреть список доступных кружков и посмотреть список своих кружков.</p>

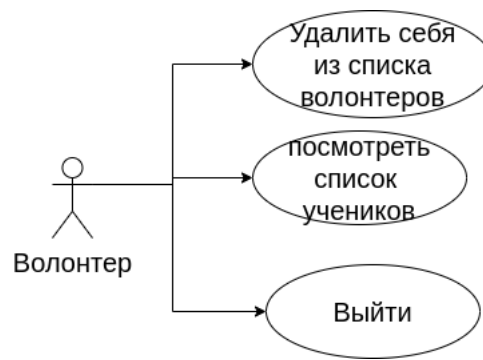


Рисунок 2 – Диаграмма прецедентов волонтера

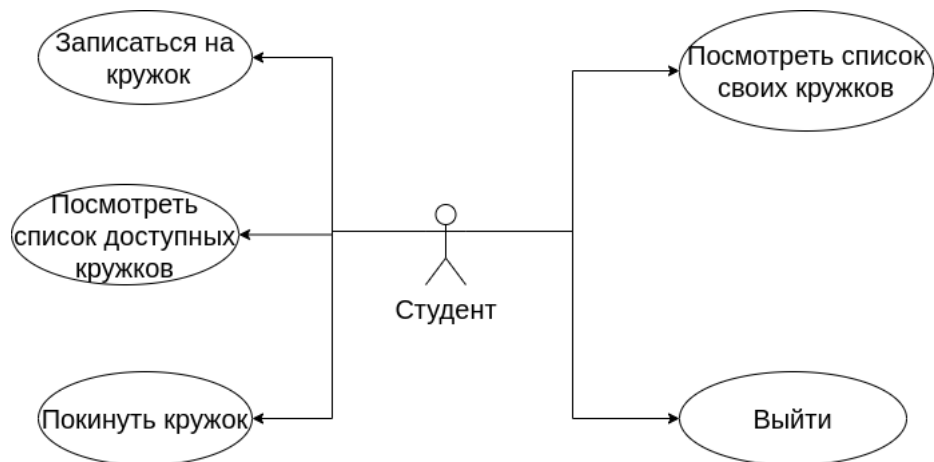


Рисунок 3 – Диаграмма прецедентов студента

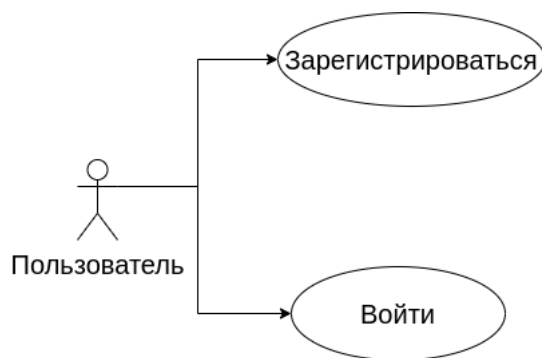


Рисунок 4 – Диаграмма прецедентов пользователя

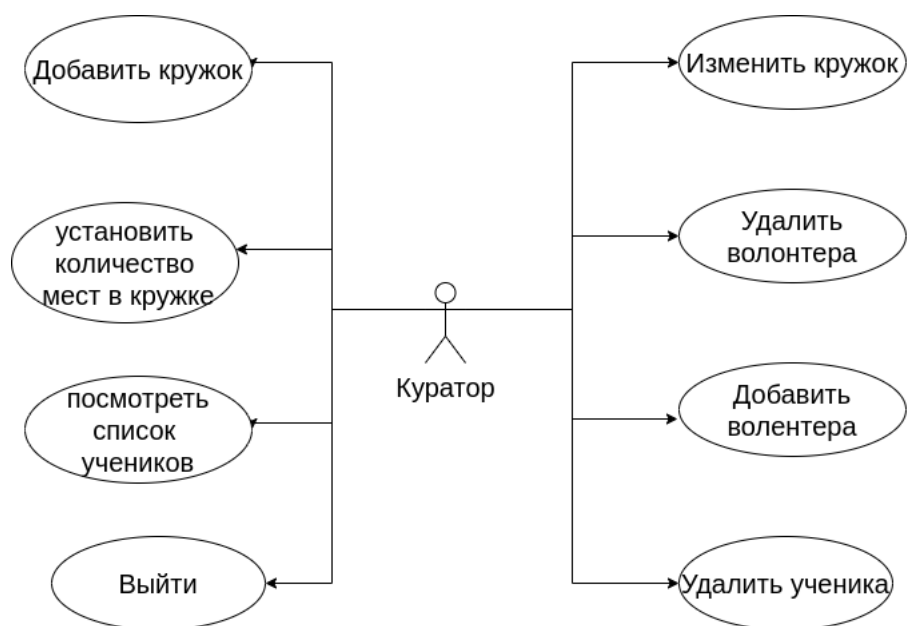


Рисунок 5 – Диаграмма прецедентов куратора

1.4 Определение понятия СУБД

Система управления базами данных (СУБД) - это комплекс программных средств, предназначенный для организации и управления базами данных [2]. СУБД обеспечивает механизмы хранения, обработки и доступа к данным, а также управления целостностью и безопасностью информации.

Основные функции СУБД включают создание и удаление баз данных, создание и удаление таблиц и отношений между ними, ввод, изменение и удаление данных, а также выполнение запросов к данным. СУБД может работать с различными моделями данных, такими как иерархическая, графовая или реляционная.

При выборе СУБД для конкретного проекта необходимо учитывать множество факторов, таких как объемы и типы данных, количество пользователей, требования к безопасности и доступности данных, а также бюджет проекта.

Таким образом, понимание понятия СУБД является важным шагом в разработке проектов по базам данных, и позволяет выбрать наиболее подходящую систему для конкретной задачи.

1.4.1 Основные функции СУБД

СУБД должна обеспечивать возможности:

- Хранение данных в удобном для работы формате;
- Обеспечение многопользовательского доступа к данным;
- Контроль целостности данных;

- Управление транзакциями;
- Обеспечение безопасности данных;
- Оптимизация запросов и индексирование данных;
- Поддержка языка запросов для извлечения информации;
- Администрирование базы данных, включая резервное копирование и восстановление данных;
- Обеспечение совместимости с другими приложениями и СУБД;
- Поддержка репликации и синхронизации данных между серверами.

1.5 Классификация СУБД по модели данных

Одной из важнейших характеристик СУБД является модель данных, которую она использует. Модель данных определяет, каким образом данные будут храниться, организовываться и манипулироваться в системе.

Существует 3 основных типа моделей организации данных:

- иерархическая;
- графовая;
- реляционная [2].

1.5.1 Иерархическая база данных

Иерархическая база данных (ИБД) (Рисунок 6) - это тип базы данных, в котором данные организованы в виде древовидной структуры, в которой каждый узел может иметь несколько дочерних узлов. В этом типе баз данных, каждый узел может иметь только одного родителя.

ИБД широко применялись в 1960-х и 1970-х годах для управления и обработки информации в научных и инженерных приложениях, а также в системах управления базами данных в банковской и финансовой сферах. Одним из наиболее известных примеров ИБД является IMS (Information Management System) от IBM [2].

Основные характеристики иерархических баз данных:

- а) Древовидная структура. Данные в ИБД организованы в виде древовидной структуры, в которой каждый узел может иметь несколько дочерних узлов.
- б) Однозначное определение родительского элемента. Каждый элемент в ИБД имеет только одного родителя;

- в) Ограниченный доступ к данным. Доступ к данным в ИБД ограничен, и запросы могут быть выполнены только для определенных узлов.
- г) Сложность обновления данных. Обновление данных в ИБД может быть сложным процессом, особенно в случаях, когда необходимо изменить структуру дерева.

Преимущества ИБД включают:

- а) Быстрый доступ к данным. ИБД обеспечивает быстрый доступ к данным благодаря их организации в древовидную структуру.
- б) Простота в использовании. ИБД просты в использовании и могут быть быстро настроены и настроены для конкретных потребностей приложения.
- в) Высокая производительность. ИБД обеспечивают высокую производительность в условиях, когда запросы выполняются только для определенных узлов.

Однако ИБД также имеют свои недостатки, включая:

- а) Ограниченные возможности для выполнения сложных запросов. ИБД не могут выполнять сложные запросы, которые могут быть выполнены с помощью других типов баз данных.
- б) Сложность обновления данных. Обновление данных в ИБД может быть сложным процессом, особенно в случаях, когда необходимо изменить структуру дерева.

Несмотря на ограничения, иерархические базы данных до сих пор используются в различных сферах, таких как банковское дело, финансы и транспортная логистика, где требуется обработка больших объемов структурированных данных. Они также широко используются в промышленности, например, для управления процессами производства и контроля качества [2][3].

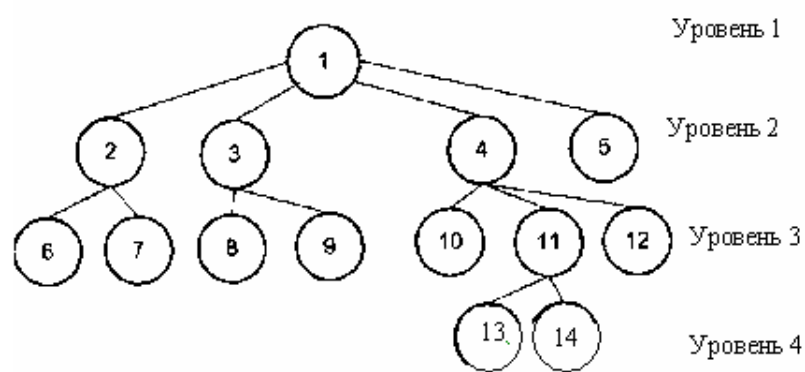


Рисунок 6 – Иерархическая база данных

1.5.2 Графовые базы данных

Графовые базы данных (ГБД) (Рисунок 7) - это тип баз данных, который использует графовую модель для представления данных и их связей [2]. Графы в ГБД состоят из узлов и ребер, которые представляют данные и связи между ними соответственно. Это позволяет представлять и хранить данные в виде графа, что очень полезно для различных типов данных, таких как социальные сети, маршруты, геоданные и другие.

Одним из преимуществ графовых баз данных является их способность обрабатывать сложные запросы, которые требуют прохождения через несколько узлов и ребер. В отличие от реляционных баз данных, графовые базы данных позволяют легко извлекать данные, которые связаны друг с другом, и выполнять сложные аналитические запросы.

Графовые базы данных также предлагают механизмы для моделирования различных типов связей между узлами, таких как направленные, ненаправленные, взвешенные, многослойные и т.д. Это позволяет легко представлять и обрабатывать различные типы данных, которые могут быть связаны между собой.

Другим важным преимуществом графовых баз данных является их способность работать с данными в реальном времени. Это позволяет обрабатывать большие объемы данных и быстро получать результаты для различных приложений, таких как мониторинг сетей, управление производственными процессами, маркетинговые исследования и т.д.

Однако, графовые базы данных имеют и свои недостатки. Они могут быть менее эффективными, чем реляционные базы данных, при работе с данными, которые не связаны между собой или имеют слабые связи. Кроме того, графовые базы данных могут быть сложными в управлении и требовать дополнительных усилий для создания и обслуживания.

В целом, графовые базы данных предоставляют мощный инструмент для работы с данными, особенно для данных, которые имеют сложную структуру и связи между ними. Они позволяют легко извлекать данные, обрабатывать и анализировать их, что делает их очень полезными для различных отраслей, таких как социальные сети, логистика, финансы, здравоохранение, и т.д.

Кроме того, в ГБД можно быстро и легко добавлять новые данные и узлы в граф, что делает их более гибкими, чем реляционные базы данных. В ГБД также можно использовать алгоритмы машинного обучения и анализа данных, что делает их еще более мощными инструментами для обработки и анализа данных[2] [3].

Некоторые примеры популярных графовых баз данных включают Neo4j, OrientDB, ArangoDB, Amazon Neptune и другие. Neo4j является одним из наиболее популярных и широко используемых ГБД в мире. Он предоставляет мощный язык запросов Cypher, который позволяет выполнять сложные запросы к графовой базе данных.

В заключение, графовые базы данных предоставляют мощный инструмент для хранения, обработки и анализа данных, особенно для данных, которые имеют сложную структуру и связи между ними. Они могут быть полезны для различных отраслей и приложений, и позволяют легко извлекать и анализировать данные в режиме реального времени.

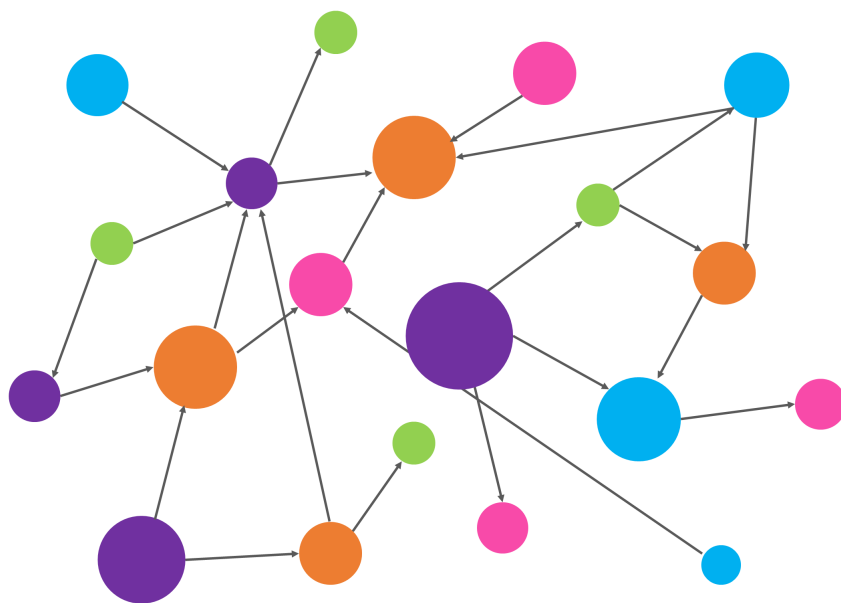


Рисунок 7 – Графовые базы данных

1.5.3 Реляционные базы данных

Реляционные базы данных (РБД) (Рисунок 8) являются одной из наиболее распространенных моделей данных. Они основаны на теории отношений и позволяют организовать данные в таблицы с помощью рядов и столбцов. Ряды (строки) представляют собой отдельные записи данных, а столбцы представляют собой атрибуты (характеристики) этих данных [2]. Таким образом, реляционная база данных состоит из одной или нескольких таблиц.

Основными преимуществами РБД являются:

- а) Простота использования: РБД легко понять и использовать благодаря своей структуре в виде таблиц. Они предоставляют простой и понятный язык для запросов данных - SQL (Structured Query Language).
- б) Гибкость: РБД позволяют быстро и легко добавлять, изменять или удалять данные без необходимости изменять структуру базы данных.
- в) Целостность данных: РБД гарантируют целостность данных благодаря механизмам проверки ограничений, что позволяет избежать ошибок и несоответствий данных.
- г) Масштабируемость: РБД могут легко масштабироваться, позволяя увеличивать объем данных и количество пользователей, работающих с данными.
- д) Поддержка транзакций: РБД обеспечивают поддержку транзакций, что позволяет сохранять целостность данных при одновременном доступе к ним нескольких пользователей.

Однако, РБД также имеют некоторые недостатки, среди которых:

- а) Сложность проектирования: Проектирование РБД может быть сложным процессом, особенно для больших и сложных систем.
- б) Ограниченность в хранении и обработке сложных объектов: РБД не могут хранить сложные объекты, такие как документы, видео или изображения, непосредственно в таблицах. Для этого нужно использовать специальные типы данных, такие как BLOB (Binary Large Object).
- в) Сложность масштабирования: При росте объемов данных и количества пользователей, работающих с базой данных, может потребоваться масштабирование системы, что может быть дорого и сложно [3].

В целом, РБД являются мощным инструментом для организации структурированных данных и обеспечения их целостности и безопасности. Они широко применяются в различных областях, таких как банковское дело, финансы, управление складами, медицина, телекоммуникации и другие.

Реляционные базы данных работают на основе нормализации данных, что означает, что данные в таблицах должны быть разделены на наименьшие возможные кусочки, чтобы минимизировать повторение информации и обеспечить целостность данных. Это также обеспечивает эффективность и быстродействие базы данных при выполнении запросов.

SQL является языком запросов, используемым для доступа к данным в РБД [3]. Он позволяет выполнять различные операции, такие как выборка данных, добавление, изменение или удаление данных, создание таблиц и другие.

Кроме того, реляционные базы данных могут быть использованы для создания отчетов и анализа данных. Они позволяют легко извлекать данные и агрегировать их, чтобы получить нужную информацию.

В целом, реляционные базы данных являются мощным и эффективным инструментом для организации структурированных данных и обеспечения их безопасности и целостности. Они позволяют эффективно хранить, обрабатывать и анализировать данные, что делает их незаменимым инструментом для многих компаний и организаций.



Рисунок 8 – Реляционные базы данных

1.6 Вывод

В данном разделе была проведена аналитическая работа, включающая в себя анализ предметной области и формализацию данных и ролей. Были рассмотрены основные типы СУБД, в том числе иерархическая, графовая и реляционная модели. Учитывая требования к разнообразным запросам и простоте использования, в качестве модели организации данных была выбрана реляционная модель. Таким образом, на данном этапе был проведен анализ и выбрана наиболее подходящая модель данных для разрабатываемого проекта.

2 Конструкторская часть

В этом разделе изучены ключевые структурные элементы баз данных, определена модель предметной области и сформулированы требования к программному обеспечению. Также выделены основные компоненты веб-приложения и описаны их функции. Для лучшего понимания некоторых методов предоставлены схемы и диаграммы.

Спроектированы функции, процедуры и триггеры, которые хранятся в базе данных. В данном разделе представлено описание их схема раз. Также предоставлены схемы и алгоритмы, поясняющие работу данных функций/процедур и триггеров.

2.1 Проектирование базы данных

База данных должна хранить рассмотренные в таблице 2 данные. В соответствии с этой таблицей можно выделить следующие таблицы:

- а) Таблица пользователей User;
- б) Таблица пользовательских данных UserInfo;
- в) Таблица ролей Role;
- г) Таблица событий Events;
- д) Таблица клубов Clubs;
- е) Таблица финансов Finances;
- ж) Таблица материалов Materials;
- з) Таблица новостей News.
- и) Развязочная таблица (пользователи - роли) – users-role.
- к) Развязочная таблица (пользовательских данных - клубов) – users-info-programs.
- л) Развязочная таблица (пользовательских данных - событий) – users-info-events.
- м) Развязочная таблица (материалов - клубов) – materials-programs.
- н) Развязочная таблица (материалов - событий) – materials-events.

Таблица users хранит информацию о пользователях.

- а) id - уникальный идентификатор пользователя: SERIAL, Primary Key.
- б) login - логин пользователя: VARCHAR(255), NOT NULL, уникальный.
- в) password - пароль пользователя: VARCHAR(255), NOT NULL.
- г) email - электронная почта пользователя: VARCHAR(100), NOT NULL.

Таблица usersInfo хранит информацию о дополнительной информации пользователей.

- а) id - уникальный идентификатор информации пользователя: SERIAL, Primary Key.
- б) user-id - уникальный идентификатор пользователя из таблицы users: SERIAL, Primary Key, Foreign Key.
- в) first-name - имя пользователя: VARCHAR(50), NOT NULL.
- г) last-name - фамилия пользователя: VARCHAR(50), NOT NULL.
- д) phone - номер телефона пользователя: VARCHAR(20).
- е) gender - пол пользователя: gender-enum.
- ж) age - возраст пользователя: INTEGER.
- з) education - уровень образования пользователя: VARCHAR(100).
- и) experience - опыт работы пользователя: VARCHAR(100).
- к) entry-date - дата создания записи: TIMESTAMP DEFAULT NOW().

Таблица role хранит информацию о ролях пользователей.

- а) id - уникальный идентификатор роли: SERIAL, Primary Key.
- б) permission - разрешение роли: user-role-enum, NOT NULL.

Таблица events хранит информацию о событиях.

- а) id - уникальный идентификатор события: SERIAL, Primary Key.
- б) name - название события: VARCHAR(100), NOT NULL.
- в) date - дата проведения события: DATE, NOT NULL.
- г) audience - целевая аудитория события: VARCHAR(100), NOT NULL.
- д) amount-of-place - количество мест для участников события: INTEGER.
- е) description - описание события: TEXT.

Таблица clubs хранит информацию о клубах.

- а) id - уникальный идентификатор клуба: SERIAL, Primary Key.
- б) name - название клуба: VARCHAR(100), NOT NULL.
- в) level - уровень клуба: club-level-enum.
- г) duration - длительность занятий клуба: INTEGER.
- д) start-of-class - время начала занятий клуба: TIME, NOT NULL.
- е) end-of-class - время окончания занятий клуба: TIME, NOT NULL.
- ж) weekday - день недели занятий клуба: weekday-enum, NOT NULL.
- з) amount-of-place - количество мест для участников клуба: INTEGER.
- и) requirements - требования к участникам клуба: TEXT.
- к) description - описание клуба: TEXT.

Таблица finances хранит информацию о финансовых операциях.

- а) id - уникальный идентификатор операции: SERIAL, Primary Key.
- б) date - дата операции: DATE, NOT NULL.
- в) description - описание операции: TEXT, NOT NULL.
- г) amount - сумма операции: NUMERIC(10, 2), NOT NULL.
- д) type - тип операции (расход или доход): financial-enum, NOT NULL.
- е) program - название программы, если операция связана с определенной программой: TEXT.
- ж) participant - имя участника, если операция связана с определенным участником: TEXT.
- з) volunteer - имя волонтера, если операция связана с определенным волонтером: TEXT.

Таблица materials хранит информацию о материалах.

- а) id - уникальный идентификатор материала: SERIAL, Primary Key.
- б) name - название материала: VARCHAR(255), not null.
- в) description - описание материала: TEXT.
- г) format - формат материала: VARCHAR(50).
- д) author - автор материала: VARCHAR(255).
- е) created-at - дата создания материала: TIMESTAMP, default now().

Таблица materials-events связывает материалы с событиями.

- а) materials-id - уникальный идентификатор материала: INTEGER, REFERENCES materials(id).
- б) event-id - уникальный идентификатор события: INTEGER, REFERENCES events(id).
- в) PRIMARY KEY (materials-id, event-id) - уникальный индекс, связывающий материал и событие.

Таблица materials-programs связывает материалы с программами.

- а) materials-id - уникальный идентификатор материала: INTEGER, REFERENCES materials(id).
- б) program-id - уникальный идентификатор программы: INTEGER, REFERENCES clubs(id).
- в) PRIMARY KEY (materials-id, program-id) - уникальный индекс, связывающий материал и программу.

Таблица news хранит информацию о новостях.

- а) id - уникальный идентификатор новости: SERIAL, Primary Key.

- б) title - заголовок новости: VARCHAR(255), not null.
- в) content - содержание новости: TEXT.
- г) date - дата публикации новости: DATE, not null.

Таблица users-info-events связывает информацию о пользователях с событиями.

- а) users-info-id - уникальный идентификатор информации о пользователе: INTEGER, REFERENCES usersInfo(id).
- б) event-id - уникальный идентификатор события: INTEGER, REFERENCES events(id).
- в) PRIMARY KEY (users-info-id, event-id) - уникальный индекс, связывающий информацию о пользователе и событие.

Таблица users-info-programs связывает информацию о пользователях с программами.

- а) users-info-id - уникальный идентификатор информации о пользователе: INTEGER, REFERENCES usersInfo(id).
- б) program-id - уникальный идентификатор программы: INTEGER, REFERENCES clubs(id).
- в) PRIMARY KEY (users-info-id, program-id) - уникальный индекс, связывающий информацию о пользователе и программу.

Таблица users-role хранит информацию о ролях пользователей.

- а) user-id - уникальный идентификатор пользователя: INTEGER, REFERENCES users(id).
- б) role-id - уникальный идентификатор роли: INTEGER, REFERENCES role(id).
- в) PRIMARY KEY (user-id, role-id) - уникальный индекс, связывающий пользователя и его роль.

Таблица news хранит новостные статьи, которые были опубликованы на сайте.

- а) id - уникальный идентификатор новостной статьи: integer, Primary Key.
- б) title - заголовок новостной статьи: varchar(255), not null.
- в) content - содержание новостной статьи: text.
- г) date - дата публикации новостной статьи: date, not null.

Схема базы данных представлена на рисунке 9.

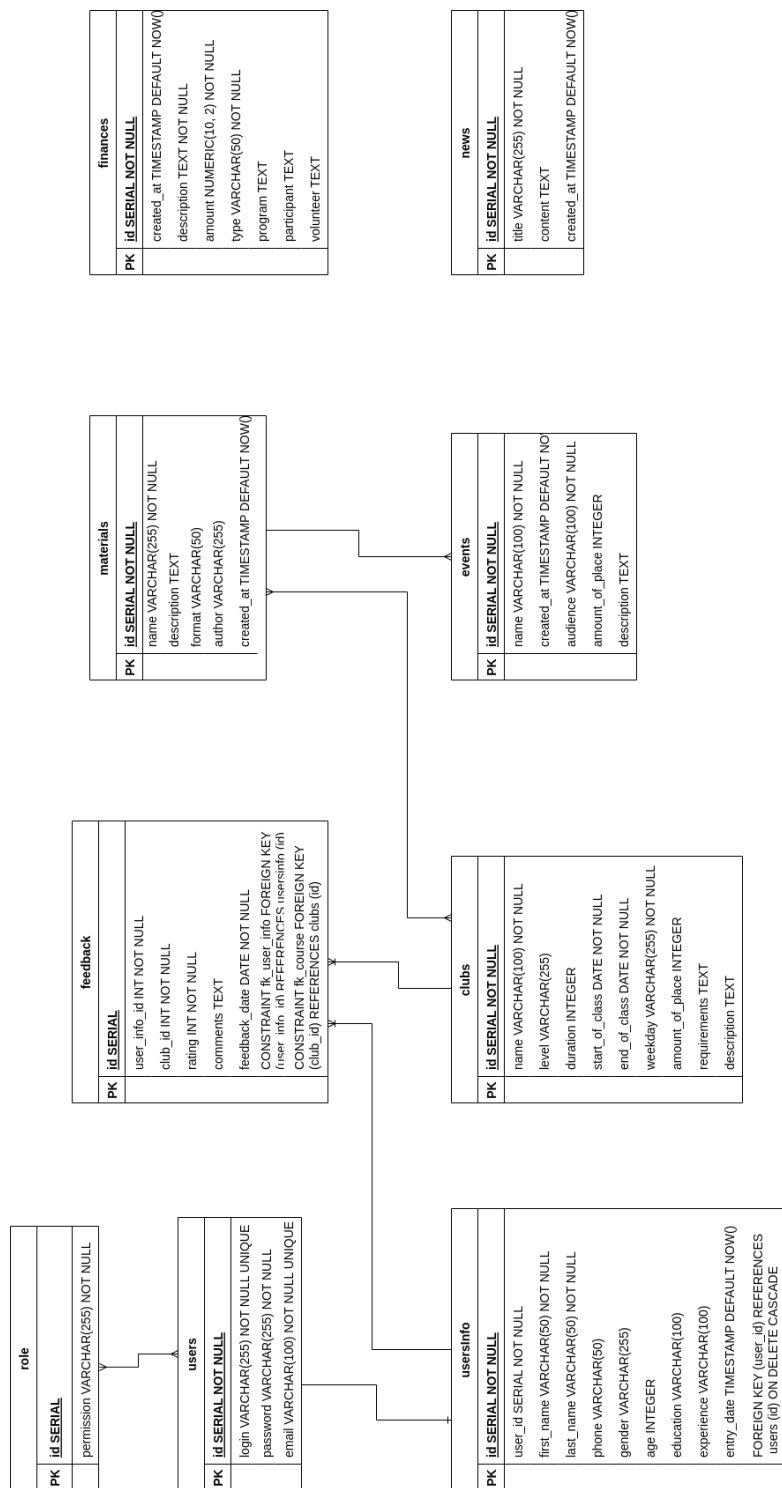


Рисунок 9 – Схема базы данных

2.2 Требования к программе

Для успешной реализации поставленной задачи программа должна предоставлять следующие возможности:

Куратор:

- Авторизоваться, зайти на страницу своего профиля;

- Авторизоваться, зайти на страницу добавить / изменить кружок и количество мест в этом кружке;
- Авторизоваться, зайти на страницу посмотреть список студентов и список студентов конкретного кружка.

Волонтер:

- Авторизоваться, зайти на страницу своего профиля;
- Авторизоваться, зайти на страницу посмотреть список студентов и список студентов конкретного кружка.

Студент:

- Авторизоваться, зайти на страницу своего профиля;
- Авторизоваться, зайти на страницу своего профиля, записаться / покинуть кружок;
- Авторизоваться, зайти на страницу посмотреть список доступных кружков и посмотреть список своих кружков.

Пользователь:

- Зайти на страницу регистрации, зарегистрироваться или перейти на страницу авторизации;
- Зайти на страницу авторизации, авторизоваться или перейти на страницу регистрации;
- Посмотреть расписание кружков;
- Авторизоваться, сделать что-то из предыдущих действий или ни одно из них, выйти из аккаунта.

2.3 Проектирование приложения

Программа, как правило, состоит из двух основных частей: front-end и back-end. Front-end – это приложение, с которым пользователь взаимодействует, включая отображение данных. Back-end – это часть программы, которая отвечает за взаимодействие с базой данных, получение, удаление и изменение данных.

Для того, чтобы следовать принципу разделения интерфейса, каждую из частей стоит выделить в отдельный проект. Кроме того, стоит отдельно добавить проект для данных, поскольку они разделяются между back-end и front-end. В результате получится следующая структура решения:

- AccessToDB – проект, отвечающий за подключение к базе данных, отправку запросов и получение информации из БД;

- DataStructures – проект, содержащий классы данных, которые дублируют структуру таблиц БД и необходимы для ослабления связанности кода и облегченного взаимодействия между БД и приложением;
- UI – проект, отвечающий за пользовательский интерфейс, через который осуществляется взаимодействие пользователя с программой.

Проект DataStructures должен состоять из четырех компонентов:

- User класс, описывающий пользователей
- UserInfo класс, описывающий пользовательские данные
- Events класс, описывающий события
- Clubs класс, описывающий клубы
- Materials класс, описывающий материалы
- Finances класс, описывающий финансовые операции
- News класс, описывающий новости

Для удовлетворения требованиям к программе, описанным в разделе 2.2, необходимы следующие функции доступа к данным:

- findAllStudentInfo() - возвращает список всех студентов;
- findAllVolunteerInfo() - возвращает список всех волонтеров (junior и senior);
- findStudentUserInfoById() - возвращает информацию о студенте с заданным ID;
- findVolunteerUserInfoById() - возвращает информацию о волонтере с заданным ID;
- findCuratorUserInfoById() - возвращает информацию о кураторе с заданным ID;
- deleteStudentInfoById() - удаляет информацию о студенте с заданным ID;
- deleteVolunteerInfoById() - удаляет информацию о волонтере с заданным ID;
- deleteAllStudentInfo() - удаляет информацию обо всех студентах;
- deleteAllVolunteerInfo() - удаляет информацию обо всех волонтерах;
- findStudentInfoByLogin() - возвращает информацию о студенте с заданным логином;
- findVolunteerInfoByLogin() - возвращает информацию о волонтере с заданным логином;

- existsByLogin() - проверяет, существует ли пользователь с заданным логином в БД;
- InsertUser() - добавление нового пользователя в БД при регистрации;
- findByName – поиск питальни по её имени;
- findClubsByAmountOfPlace – поиск питальни, которая имеет определенное количество мест;
- findFinancesByProgram – поиск финансовой информации по программе;
- existsEventByName – проверка существования мероприятия по имени;
- findEventByName – поиск мероприятия по имени;
- findAllUserEvents – поиск всех мероприятий, на которые зарегистрирован пользователь;
- existsByName - возвращает true, если в таблице Material существует запись с заданным именем, иначе - false;
- findByName - возвращает Optional объект Material с заданным именем или пустой Optional, если такой записи нет в таблице;
- findByFormat - возвращает список объектов Material, которые имеют заданный формат;
- findByTitle - возвращает список объектов News с заданным заголовком;
- findByDate - возвращает список объектов News, созданных в заданную дату;
- existsByTitle - возвращает true, если в таблице News существует запись с заданным заголовком, иначе - false.

Схема разработанного приложения представлена на рисунке 10 - 11.

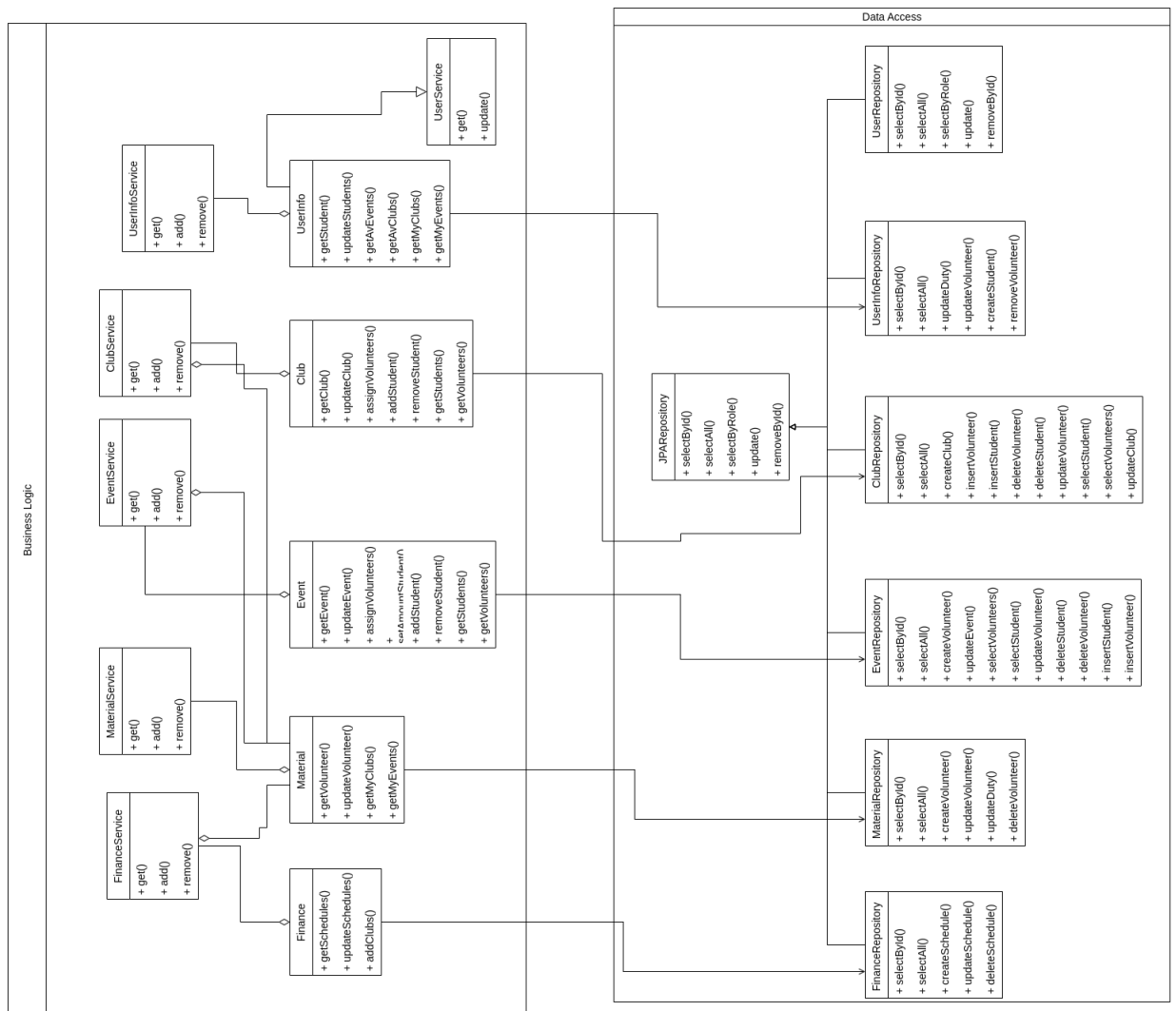


Рисунок 10 – Схема разработанного back-end

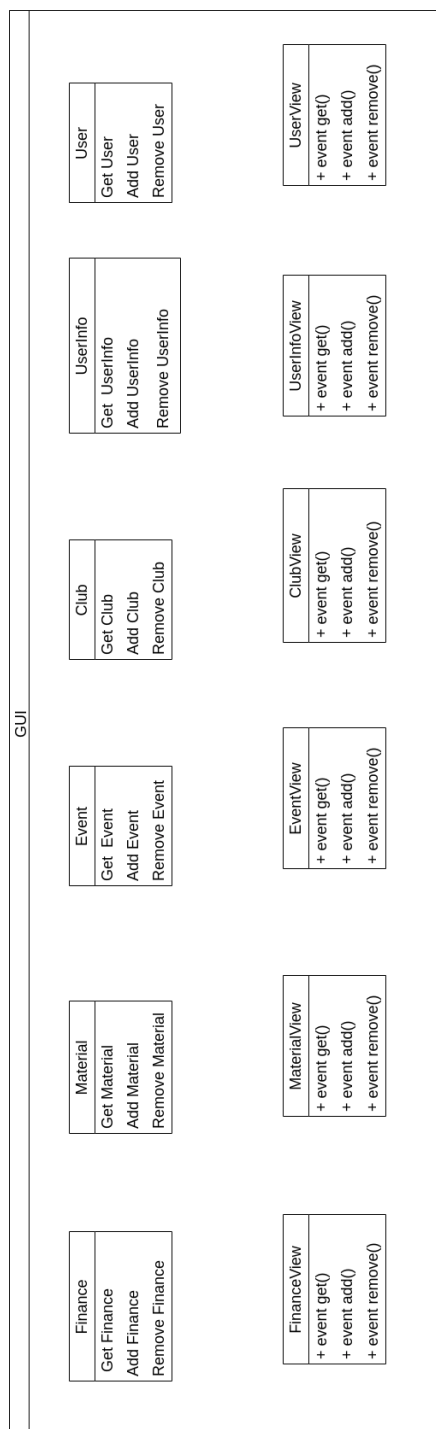


Рисунок 11 – Схема разработанного front-end

2.4 Проектирование хранимых в БД функции/процедур

Для обеспечения удобного и эффективного доступа к данным, в базе данных были разработаны и реализованы следующие функции:

Функция `get-yearly-finances-statistic` принимает аргумент `type-of-finance` типа `varchar` и возвращает таблицу с тремя столбцами: `type` (тип финансов), `month` (месяц) и `total-amount` (общая сумма). Эта функция использует данные из таблицы `finances`.

Функция `get-active-user-statistics` принимает аргумент `user-role` типа `varchar` и возвращает таблицу с четырьмя столбцами: `first-name` (имя), `last-name` (фамилия), `event-count` (количество событий) и `club-count` (количество клубов). Эта функция использует данные из нескольких таблиц: `usersInfo`, `users-info-events`, `events`, `users-info-programs`, `clubs`, `users`, `users-role` и `role`. Функция связывает эти таблицы по различным связям и фильтрует результаты по значению `user-role` в поле `permission` таблицы `role`.

2.5 Проектирование хранимых в БД триггеров

База данных должна содержать следующий триггер:

Триггер `on-delete-userInfo` (Рисунок 12):

Функция `curator-safety-delete-userInfo()` проверяет роль пользователя, пытающегося удалить запись в таблице `usersinfo`. Если роль пользователя является `Curator`, выбрасывается исключение с сообщением `"Deletion of UserInfo Curator is not allowed."` Триггер `on-delete-userInfo` срабатывает перед операциями `DELETE` или `INSERT` на таблице `usersinfo` для каждой обрабатываемой строки. Он выполняет функцию `curator-safety-delete-userInfo()`.

Эти триггер обеспечивают безопасность и ограничивают возможность удаления пользователей и информации о пользователях с ролью `Curator`. В случае попытки удаления таких записей, соответствующие триггеры выбрасывают исключения, предотвращая удаление.

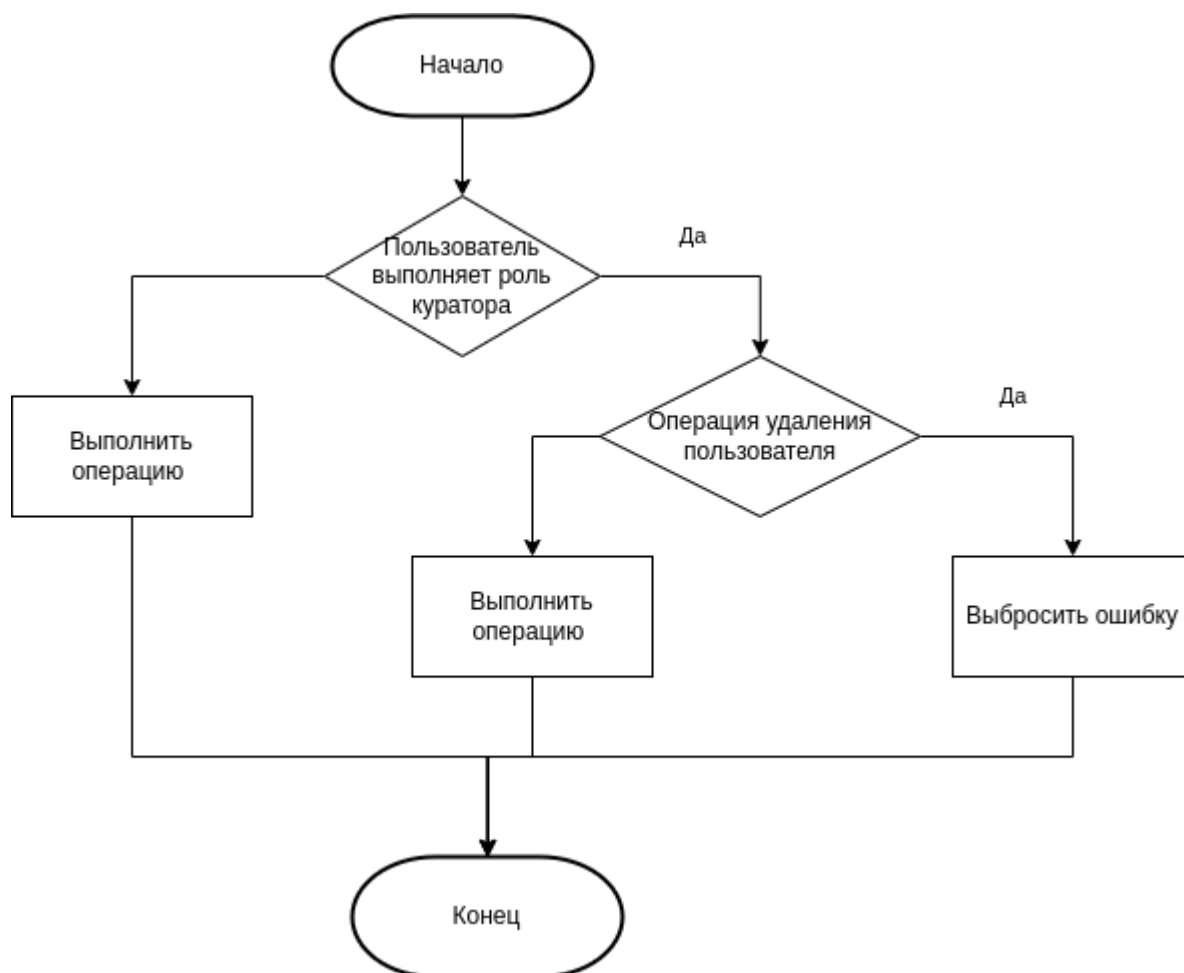


Рисунок 12 – Схема триггера на удаление записей пользователя с ролью куратора

2.6 Вывод

В данном разделе были рассмотрены основные структурные элементы базы данных и определена модель предметной области. Также были сформулированы требования к программе и выделены основные компоненты веб-приложения с их функционалом. В дополнение были приведены схемы некоторых методов для упрощения их понимания и использования в дальнейшей работе.

3 Технологическая часть

В рамках технологической составляющей обоснован выбор языка программирования, среды разработки, системы управления базами данных (СУБД) и дополнительных технологий. Представлены структура классов программы и отрывки кода. Кроме того, продемонстрированы возможности веб-интерфейса.

3.1 Выбор и обоснование используемых технологий

В рамках технической части курсовой работы были выбраны следующие технологии: Java, Spring Boot, Spring Data, Spring MVC, Spring Security и Hibernate для разработки бэкенда системы. Для фронтенда были выбраны TypeScript, React и Redux. Для управления базой данных был выбран PostgreSQL.

Java является языком программирования, используемым для разработки бэкенд-приложений. Он известен своей надежностью, высокой производительностью и расширяемостью. Spring Boot, в свою очередь, предоставляет удобный и эффективный фреймворк для создания самостоятельных, самоуправляемых и легко масштабируемых Java-приложений.

Spring Data является частью Spring Framework и предоставляет абстракцию для работы с базами данных. Он значительно упрощает и ускоряет разработку, предоставляя удобные аннотации и методы для выполнения операций с данными.

Spring MVC (Model-View-Controller) является фреймворком, который обеспечивает архитектурный шаблон MVC для разработки веб-приложений. Он разделяет логику приложения на три компонента: модель (Model), представление (View) и контроллер (Controller), что облегчает разработку и поддержку приложений.

Spring Security предоставляет механизмы аутентификации и авторизации для Java-приложений. Он обеспечивает защиту приложения от несанкционированного доступа и обеспечивает безопасность данных.

Hibernate - это фреймворк для объектно-реляционного отображения (ORM), который упрощает взаимодействие с базой данных, позволяя работать с объектами вместо прямых SQL-запросов. Он автоматически обрабатывает маппинг объектов на таблицы базы данных и упрощает выполнение операций с данными.

В фронтенд-разработке использовались TypeScript, React и Redux. TypeScript - это язык программирования, который добавляет статическую типизацию к JavaScript и облегчает разработку сложных веб-приложений. React - это JavaScript-библиотека для создания пользовательских интерфейсов, которая позволяет разделять интерфейс на компоненты и обеспечивает эффективное обновление пользовательского интерфейса. Redux - это предсказуемое состояние приложения, которое упрощает управление состоянием приложения и обеспечивает единообразный поток данных.

Также был использован Liquibase - фреймворк для управления миграциями базы данных. Он позволяет управлять изменениями схемы базы данных и обеспечивает контроль версий структуры данных[4].

PostgreSQL - мощную и надежную реляционную СУБД. Она обладает расширенными возможностями по обработке структурированных данных и хранению информации[1].

В качестве среды разработки был использован IntelliJ. Это мощное интегрированное средство разработки (IDE) для Java, которое предоставляет удобный интерфейс и полезные функции, такие как автодополнение кода, отладка и управление версиями [5].

Выбор данных технологий был обусловлен их широкой популярностью, хорошей документацией и поддержкой сообщества разработчиков. Они также обладают необходимыми функциональными возможностями для реализации поставленных целей курсовой работы.

3.2 Создание объектов БД

3.2.1 Создание таблиц

Таблица users

Ниже, на листинге 1, представлено создание таблицы Users. Поле id является уникальным идентификатором пользователя, поэтому на него накладывается ограничение Primary Key. На поля login, password и email также накладывается ограничение not null, что означает, что эти поля должны содержать значения и не могут быть пустыми. Ограничение unique на поле login гарантирует уникальность значений в этом поле, а ограничение unique на поле email обеспечивает уникальность значений в поле email.

Листинг 1 – Создание таблицы Users.

```
CREATE TABLE users
(
    id          SERIAL PRIMARY KEY NOT NULL ,
    login       VARCHAR(255) NOT NULL unique ,
    password    VARCHAR(255) NOT NULL ,
    email       VARCHAR(100) NOT NULL unique
);
```

Таблица usersInfo

Ниже, на листинге 2, представлено создание таблицы UsersInfo. Поле id является уникальным идентификатором записи в таблице, поэтому на него накладывается ограничение Primary Key. Поле user-id связано с полем id таблицы users посредством внешнего ключа FOREIGN KEY (user-id) REFERENCES users (id) ON DELETE CASCADE, что обеспечивает целостность данных между этими таблицами. Остальные поля, такие как first-name, last-name, phone, gender, age, education, experience и entry-date, не имеют специальных ограничений, но могут содержать значения или быть пустыми в зависимости от конкретной ситуации.

Листинг 2 – Создание таблицы UsersInfo.

```
CREATE TABLE usersInfo
(
    id SERIAL PRIMARY KEY NOT NULL ,
    user_id SERIAL NOT NULL ,
    first_name VARCHAR(50) NOT NULL ,
    last_name  VARCHAR(50) NOT NULL ,
    phone      VARCHAR(50) ,
    gender     varchar(255) ,
    age        INTEGER ,
    education  VARCHAR(100) ,
    experience VARCHAR(100) ,
    entry_date TIMESTAMP DEFAULT NOW() ,
    FOREIGN KEY (user_id) REFERENCES users (id) ON DELETE CASCADE
);
```

Таблица role

Ниже, на листинге 3, представлено создание таблицы Role. Поле id является уникальным идентификатором роли, поэтому на него накладывается ограничение Primary Key. Поле permission представляет собой разрешение для роли и должно содержать значения, поэтому на него накладывается ограничение not null.

Листинг 3 – Создание таблицы Role.

```
CREATE TABLE role
(
    id          serial primary key,
    permission varchar(255) NOT NULL
);
```

Таблица events

Ниже, на листинге 4, представлено создание таблицы Events. Поле id является уникальным идентификатором события, поэтому на него накладывается ограничение Primary Key. Поля name и audience должны содержать значения и не могут быть пустыми, поэтому на них накладывается ограничение not null. Поле created-at представляет дату создания события и имеет значение по умолчанию NOW(), что означает текущую дату и время. Поле amount-of-place представляет количество доступных мест для события, а поле description содержит текстовое описание события и может быть пустым или содержать текстовую информацию.

Листинг 4 – Создание таблицы Events.

```
CREATE TABLE events
(
    id          SERIAL PRIMARY KEY NOT NULL ,
    name        VARCHAR(100) NOT NULL ,
    created_at   TIMESTAMP DEFAULT NOW() ,
    audience     VARCHAR(100) NOT NULL ,
    amount_of_place INTEGER ,
    description   TEXT
);
```

Таблица clubs

Ниже, на листинге 5, представлено создание таблицы Clubs. Поле id является уникальным идентификатором клуба, поэтому на него накладывается ограничение Primary Key. Поле name представляет название клуба и должно содержать значения, поэтому на него накладывается ограничение not null. Поля start-of-class и end-of-class представляют дату начала и окончания занятий клуба соответственно, и они не могут быть пустыми, поэтому на них также накладывается ограничение not null. Поле weekday указывает день недели, в который проводятся занятия клуба. Поля level, duration, amount-of-place, requirements и description могут содержать текстовую информацию и быть пустыми в зависимости от конкретной ситуации.

Листинг 5 – Создание таблицы Clubs.

```
CREATE TABLE clubs
(
    id            SERIAL PRIMARY KEY NOT NULL ,
    name          VARCHAR(100) NOT NULL ,
    level         varchar(255),
    duration      INTEGER ,
    start_of_class date          NOT NULL ,
    end_of_class  date          NOT NULL ,
    weekday       varchar(255) NOT NULL ,
    amount_of_place INTEGER ,
    requirements  TEXT ,
    description   TEXT
);
```

Таблица finances

Ниже, на листинге 6, представлено создание таблицы Finances. Поле id является уникальным идентификатором финансовой операции, поэтому на него накладывается ограничение Primary Key. Поле created-at представляет дату создания операции и имеет значение по умолчанию NOW(), что означает текущую дату и время. Поле description содержит описание операции и не может быть пустым. Поле amount представляет сумму операции и имеет тип NUMERIC(10, 2), что означает, что значение может содержать до 10 цифр, с 2 знаками после запятой. Поле type указывает тип операции, например, income или expense. Поля program, participant и volunteer могут содержать текстовую информацию и быть пустыми в зависимости от конкретной ситуации.

Листинг 6 – Создание таблицы Finances.

```
CREATE TABLE finances
(
    id            SERIAL PRIMARY KEY NOT NULL ,
    created_at    TIMESTAMP DEFAULT NOW() ,
    description   TEXT          NOT NULL ,
    amount        NUMERIC(10, 2) NOT NULL ,
    type          varchar(50) NOT NULL ,
    program       TEXT ,
    participant   TEXT ,
    volunteer     TEXT
);
```

Ниже, на листинге 7, представлено создание таблицы News. Эта таблица предназначена для хранения новостей. Она содержит следующие поля:

Поле id имеет тип SERIAL и является первичным ключом таблицы. Это поле представляет уникальный идентификатор новости. Поле title имеет тип VARCHAR(255) и предназначено для хранения заголовка новости. Оно обязательно для заполнения (NOT NULL). Поле content имеет тип TEXT и предназначено для хранения содержимого новости. Данное поле может содержать более длинные тексты и не является обязательным для заполнения (NULL). Поле created-at имеет тип TIMESTAMP и устанавливает время создания новости. При создании новой записи в этой таблице, если не указано явно, оно будет автоматически установлено на текущее время с помощью значения DEFAULT NOW().

Листинг 7 – Создание таблицы News.

```
CREATE TABLE news
(
    id          SERIAL PRIMARY KEY NOT NULL ,
    title       VARCHAR(255) NOT NULL ,
    content     TEXT ,
    created_at  TIMESTAMP DEFAULT NOW()
);
```

Таблица materials

Ниже, на листинге 8, представлено создание таблицы Materials. Поле id является уникальным идентификатором материала, поэтому на него накладывается ограничение Primary Key. Поле name представляет название материала и не может быть пустым. Поле description содержит описание материала и может быть пустым. Поле format указывает формат материала, например, PDF или MP4. Поле author представляет автора материала. Поле created-at представляет дату создания материала и имеет значение по умолчанию NOW(), что означает текущую дату и время.

Листинг 8 – Создание таблицы Materials.

```
CREATE TABLE materials
(
    id          SERIAL PRIMARY KEY NOT NULL ,
    name        VARCHAR(255) NOT NULL ,
    description  TEXT ,
    format       VARCHAR(50) ,
    author       VARCHAR(255) ,
    created_at  TIMESTAMP DEFAULT NOW());
```

3.2.2 Создание связей

Таблица materials-events

Ниже, на листинге 9, представлено создание таблицы Materials-Events. Эта таблица является связующей таблицей между материалами и событиями. Поле materials-id является внешним ключом, которое ссылается на поле id в таблице Materials. Поле event-id является внешним ключом, которое ссылается на поле id в таблице Events. Эта таблица использует составной первичный ключ, объединяющий оба поля materials-id и event-id, чтобы обеспечить уникальность комбинации материала и события.

Листинг 9 – Создание таблицы Materials-Events.

```
CREATE TABLE materials_events
(
    materials_id INTEGER REFERENCES materials (id),
    event_id      INTEGER REFERENCES events (id),
    PRIMARY KEY (materials_id, event_id)
);
```

Таблица materials-programs

Ниже, на листинге 10, представлено создание таблицы Materials-Programs. Эта таблица является связующей таблицей между материалами и программами (клубами). Поле materials-id является внешним ключом, которое ссылается на поле id в таблице Materials. Поле program-id является внешним ключом, которое ссылается на поле id в таблице Clubs. Эта таблица также использует составной первичный ключ, объединяющий оба поля materials-id и program-id, чтобы обеспечить уникальность комбинации материала и программы.

Листинг 10 – Создание таблицы Materials-Programs.

```
CREATE TABLE materials_programs
(
    materials_id INTEGER REFERENCES materials (id),
    program_id   INTEGER REFERENCES clubs (id),
    PRIMARY KEY (materials_id, program_id));
```

Таблица users-info-events

Ниже, на листинге 11, представлено создание таблицы Users-Info-Events. Эта таблица является связующей таблицей между информацией о пользователях и событиях. Поле users-info-id является внешним ключом, которое ссылается на поле id в таблице Users-Info. Поле event-id является внешним ключом, которое ссылается на поле id в таблице Events. Эта таблица также использует составной первичный ключ, объединяющий оба поля users-info-id и event-id, чтобы обеспечить уникальность комбинации информации о пользователе и события.

Листинг 11 – Создание таблицы Users-Info-Events.

```
CREATE TABLE users_info_events
(
    users_info_id INTEGER REFERENCES usersInfo (id),
    event_id INTEGER REFERENCES events (id),
    PRIMARY KEY (users_info_id, event_id)
);
```

Таблица users-info-programs

Ниже, на листинге 12, представлено создание таблицы Users-Info-Programs. Эта таблица является связующей таблицей между информацией о пользователях и программами (клубами). Поле users-info-id является внешним ключом, которое ссылается на поле id в таблице Users-Info. Поле program-id является внешним ключом, которое ссылается на поле id в таблице Clubs. Эта таблица также использует составной первичный ключ, объединяющий оба поля users-info-id и program-id, чтобы обеспечить уникальность комбинации информации о пользователе и программы.

Листинг 12 – Создание таблицы Users-Info-Programs.

```
CREATE TABLE users_info_programs
(
    users_info_id INTEGER REFERENCES usersInfo (id),
    program_id INTEGER REFERENCES clubs (id),
    PRIMARY KEY (users_info_id, program_id)
);
```

Таблица users-role

Ниже, на листинге 13, представлено создание таблицы Users-Role. Эта таблица является связующей таблицей между пользователями и ролями. Поле users-id является внешним ключом, которое ссылается на поле id в таблице Users. Поле role-id является внешним ключом, которое ссылается на поле id в таблице Role. Эта таблица также использует составной первичный ключ, объединяющий оба поля users-id и role-id, чтобы обеспечить уникальность комбинации пользователя и роли.

Листинг 13 – Создание таблицы Users-Role.

```
CREATE TABLE users_role
(
    users_id INTEGER REFERENCES users (id),
    role_id INTEGER REFERENCES role (id),
    PRIMARY KEY (users_id, role_id)
);
```

3.2.3 Создание ролей и выделение им прав

В конструкторском разделе была разработана ролевая модель, которая включает следующие роли: студент (учащийся), волонтер, куратор. Сценарий создания ролей и назначения им прав представлен в листинге 17.

Листинг 14 – Создание ролей и выделение им прав.

```
CREATE ROLE admin WITH
    SUPERUSER
    NOCREATEDB
    CREATEROLE
    NOINHERIT
    NOREPLICATION
    NOBYPASSRLS
    CONNECTION LIMIT -1
    LOGIN
    PASSWORD 'admin';

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

CREATE ROLE _volunteer WITH
```

Листинг 15 – Продолжение создание ролей и выделение им прав.

```
NOSUPERUSER
NOCREATEDB
NOCREATEROLE
NOINHERIT
NOREPLICATION
NOBYPASSRLS
CONNECTION LIMIT -1
LOGIN
PASSWORD 'volunteer';
```

GRANT SELECT

```
ON public."clubs",
public."events",
public."materials",
public."news",
public."usersinfo",
public."users_info_events",
public."users_info_programs",
public."materials_events",
public."materials_programs",
public."feedback",
public."users_role"
TO _volunteer;
```

GRANT INSERT

```
ON public."usersinfo",
public."feedback"
TO _volunteer;
```

GRANT DELETE

```
ON public."usersinfo",
public."users",
public."users_role",
public."feedback"
TO _volunteer;
```

GRANT UPDATE

```
ON public."usersinfo",
public."users",
```

Листинг 16 – Продолжение создание ролей и выделение им прав.

```
public."feedback"
TO _volunteer;

CREATE ROLE _student WITH
    NOSUPERUSER
    NOCREATEDB
    NOCREATEROLE
    NOINHERIT
    NOREPLICATION
    NOBYPASSRLS
    CONNECTION LIMIT -1
    LOGIN
    PASSWORD 'student';

GRANT SELECT
    ON public."clubs",
    public."events",
    public."materials",
    public."news",
    public."usersinfo",
    public."users_info_events",
    public."users_info_programs",
    public."materials_events",
    public."materials_programs",
    public."feedback",
    public."users_role"
TO _student;

GRANT INSERT
    ON public."usersinfo",
    public."users_info_events",
    public."users_info_programs",
    public."feedback"
TO _student;

GRANT DELETE
    ON public."usersinfo",
    public."users",
```

Листинг 17 – Продолжение создание ролей и выделение им прав.

```
public."users_role",
public."users_info_events",
public."users_info_programs",
public."feedback"
TO _student;

GRANT UPDATE
ON public."usersinfo",
public."users",
public."users_info_events",
public."users_info_programs",
public."feedback"
TO _student;

CREATE ROLE guest WITH
NOSUPERUSER
NOCREATEDB
NOCREATEROLE
NOINHERIT
NOREPLICATION
NOBYPASSRLS
CONNECTION LIMIT -1
LOGIN
PASSWORD 'guest';

GRANT SELECT
ON public."clubs",
public."events",
public."feedback",
public."news"
TO guest;

GRANT INSERT
ON public."users",
public."users_role"
TO guest;
```

3.2.4 Создание функций

Функция **get-yearly-finances-statistic(type-of-finance varchar)**

Эта функция возвращает статистику годовых финансовых показателей для определенного типа финансовых операций. Функция принимает параметр `type-of-finance`, который указывает тип финансовых операций, для которых требуется получить статистику. Функция возвращает таблицу с тремя столбцами:

`type` тип `text`: тип финансовых операций. `month` тип `timestamp`: месяц и год, для которых рассчитывается статистика. `total-amount` тип `numeric`: общая сумма финансовых операций за указанный месяц и год. Функция выполняет следующие действия:

Выполняет запрос к таблице `finances`, используя параметр `type-of-finance` для фильтрации по указанному типу финансовых операций. Группирует результаты запроса по типу операций и месяцу, используя функцию `date-trunc` для округления даты до начала месяца. Вычисляет сумму финансовых операций для каждого типа и месяца. Сортирует результаты по месяцу в возрастающем порядке.

Листинг 18 – Создание функции `get-yearly-finances-statistic`.

```
CREATE OR REPLACE FUNCTION get_yearly_finances_statistic(  
    type_of_finance varchar)  
RETURNS TABLE (  
    type text,  
    month timestamp,  
    total_amount numeric  
) AS $$  
BEGIN  
    RETURN QUERY  
        SELECT type, date_trunc('month', date) as month, SUM(amount) as  
            total_amount  
        FROM finances  
        WHERE date >= DATE_TRUNC('year', CURRENT_DATE) and type =  
            type_of_finance  
        GROUP BY type, month  
        ORDER BY month;  
END;  
$$ LANGUAGE plpgsql;
```

Функция **get-active-user-statistics(user-role varchar)**

Эта функция возвращает статистику активных пользователей в системе для определенной роли пользователя. Функция принимает параметр user-role, который указывает роль пользователей, для которых требуется получить статистику. Функция возвращает таблицу с четырьмя столбцами:

first-name тип text: имя активного пользователя. last-name тип text: фамилия активного пользователя. event-count тип bigint: количество событий (event), в которых участвовал пользователь. club-count тип bigint: количество клубов (club), в которых участвовал пользователь. Функция выполняет следующие действия:

Выполняет запрос, объединяющий таблицы usersInfo, users-info-events, events, users-info-programs и clubs для получения информации об активных пользователях, связанных с событиями и клубами. Группирует результаты запроса по идентификатору пользователя. Вычисляет количество уникальных событий (event) и клубов (club), в которых участвовал каждый пользователь, с помощью функции COUNT(DISTINCT). Фильтрует результаты запроса по указанной роли пользователя. Сортирует результаты по количеству событий (event) в убывающем порядке, а затем по количеству клубов (club) в убывающем порядке.

Листинг 19 – Создание функции get-active-user-statistics.

```
CREATE OR REPLACE FUNCTION get_active_user_statistics(user_role
    varchar)
RETURNS TABLE (
    first_name text,
    last_name text,
    event_count bigint,
    club_count bigint
) AS $$
BEGIN
    RETURN QUERY
        SELECT ui.first_name, ui.last_name, COUNT(DISTINCT e.id) AS
            event_count, COUNT(DISTINCT c.id) AS club_count
        FROM usersInfo ui
        LEFT JOIN users_info_events uie ON ui.id = uie.users_info_id
        LEFT JOIN events e ON uie.event_id = e.id
        LEFT JOIN users_info_programs uip ON ui.id = uip.users_info_id
        LEFT JOIN clubs c ON uip.program_id = c.id
        JOIN users u on u.id = ui.user_id
        JOIN users_role ur on u.id = ur.users_id
        JOIN role r on r.id = ur.role_id
        where permission = user_role
```

Листинг 20 – Продолжение создание функции get-active-user-statistics.

```
GROUP BY ui.id
ORDER BY event_count DESC, club_count DESC;
END;
$$ LANGUAGE plpgsql;
```

3.2.5 Создание триггеров

Функция curator-safety-delete-userInfo()

Эта функция является триггерной функцией, которая выполняется перед удалением или вставкой записей в таблицу usersinfo. Функция предотвращает удаление информации о пользователе (usersinfo) для пользователей с ролью CURATOR (куратор). Функция возвращает объект типа trigger.

Функция выполняет следующие действия:

Объявляет переменную permission типа cstring, в которую будет сохранено значение разрешения роли пользователя. Выполняет запрос, который извлекает значение разрешения роли пользователя из таблицы role. Значение сохраняется в переменной permission. Запрос соединяет таблицы role, users-role, users и usersinfo, чтобы получить разрешение роли для пользователя, на которого ссылаются старые данные (OLD). Если разрешение роли равно CURATOR, то генерируется исключение с сообщением "Deletion of UserInfo Curator is not allowed."

Листинг 21 – Создание триггера on-delete-userInfo.

```
CREATE FUNCTION curator_safety_delete_userInfo()
  RETURNS trigger AS
$$
DECLARE
  permission cstring;
BEGIN
  SELECT role.permission
  INTO permission
  FROM role
        JOIN users_role ur on role.id = ur.role_id
        join users u on u.id = ur.users_id
        join usersinfo u2 on u.id = u2.user_id;
  IF permission = 'CURATOR' THEN
    RAISE EXCEPTION 'Deletion of UserInfo Curator is not allowed
    .';
```


Листинг 22 – Продолжение создание триггера on-delete-userInfo.

```
END IF;  
RETURN OLD;  
END;  
$$ LANGUAGE plpgsql;  
CREATE TRIGGER on_delete_userInfo  
    BEFORE DELETE or INSERT  
    ON usersinfo  
    FOR EACH ROW  
EXECUTE FUNCTION curator_safety_delete_userInfo();
```

3.3 Графический интерфейс

Пользовательский интерфейс веб-приложения разрабатывается на клиентской стороне с использованием Typescript, React, Redux включает в себя несколько веб-страниц.

На главной веб-странице для неавторизованного пользователя (отслеживаемой по URL <http://127.0.0.1:5000/>) представлена основная информация о функционале приложения. В верхней части экрана доступна меню (Рисунок 13).

Для выполнения процесса авторизации необходимо предоставить правильный логин и пароль уже зарегистрированного пользователя в системе (Рисунок 14).

Для регистрации нового пользователя необходимо выполнить следующие шаги.

Перейдя по ссылке "Sign Up" (Рисунок 15). На первой веб-странице нужно ввести email-адрес, пароль и другую необходимую информацию. Обработчики запросов выполняют проверку введенных данных на корректность и их соответствие требованиям системы. Добавление пользователя в базу данных происходит только при успешном вводе данных.

После входа в личный кабинет пользователь попадает в свой профиль (Рисунок 16), где отображается информация о нем и предоставляется возможность ее изменения. Для всех пользователей доступно меню, в котором они могут просмотреть список новостей (Рисунок 17), клубов, событий и материалов. Для студентов, помимо обычного меню, предусмотрена возможность просмотра своих зарегистрированных клубов и событий (Рисунок 19), а также регистрации на другие клубы и события (Рисунок 20). Для кураторов появляются дополнительные веб-

страницы, где они могут создавать различные сущности, за исключением студентов (Рисунок 18). Кураторы также имеют возможность изменять и удалять сущности (Рисунок 21). Кроме всех перечисленных возможностей, студенты имеют возможность оставлять отзывы о волонтерах и мероприятиях (Рисунок 22), которые происходят в волонтерской организации.

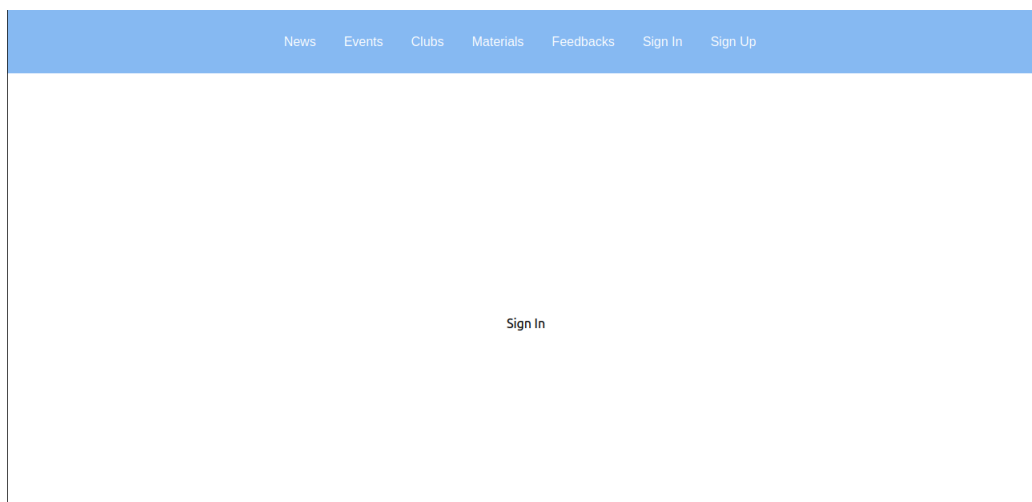


Рисунок 13 – Главная веб-страница

The image displays a login form on a light gray background. The form is titled 'Sign In' in a large, bold, black font. Below the title, there are two labels: 'Login:' and 'Password:', both in bold black font. Each label is followed by a white rectangular input field with a thin gray border. At the bottom of the form, there is a green rectangular button with rounded corners and the text 'Sign In' in white.

Рисунок 14 – Веб-страница для авторизации

First Name:

Last Name:

Phone:

Gender:

Male ▾

Age:

18 ▴ ▾

Education:

Experience:

Login:

Password:

Email:

Register

Рисунок 15 – Веб-страница для регистрации

User Information

First Name: ilhom

Last Name: nazirov

Phone: +79250403512

Gender: MALE

Age: 18

Education: dsfsdfsdfsdf

Experience: sdffsdfsdf

Login: nazirov

Password: *****

Email: naziroffjr@gmail.com

Edit

Рисунок 16 – Веб-страница профиль пользователя


zczsdasdasdas dsafsadfsdfsdfasfsadfasdfadsfasdfsdfsdf	werwer sdfsdfsdfsdfsdfsdfsdf	cxvcxvvxcv xcvxzvxcfsafsfsafsfvzvzxdsfas
wasd asdadasddas	ilhom zdssdzsds	wwwwwwwwww ssssss

Рисунок 17 – Веб-страница список новостей

Create Event


Event Name:

Event Date:




Audience:

Amount of Place:



Description:



Create Event

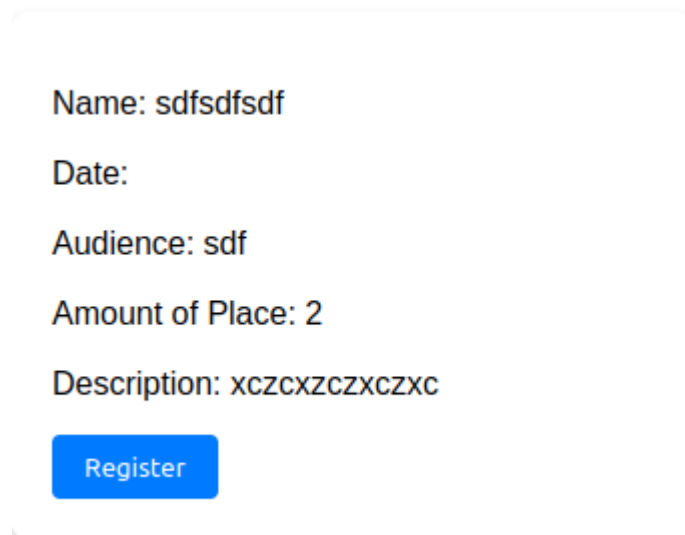
Рисунок 18 – Веб-страница для создания событий

asdasd

MONDAY

asdssasdasdsa

Рисунок 19 – Веб-страница для просмотра клубов пользователя



A registration form with a light gray background and rounded corners. It contains five text labels with placeholder text, followed by a blue 'Register' button.

Name: sdfsd sdf

Date:

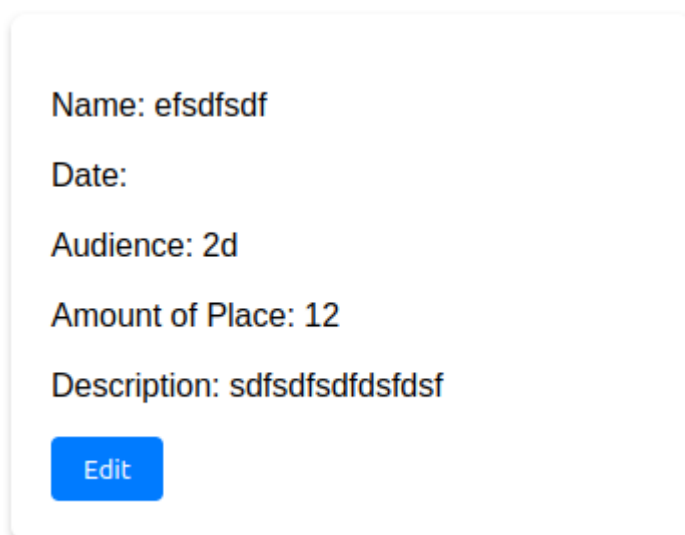
Audience: sdf

Amount of Place: 2

Description: xcxcxcxcxcxc

Register

Рисунок 20 – Веб-страница для регистрации на события



An edit event form with a light gray background and rounded corners. It contains five text labels with placeholder text, followed by a blue 'Edit' button.

Name: efsd sdf

Date:

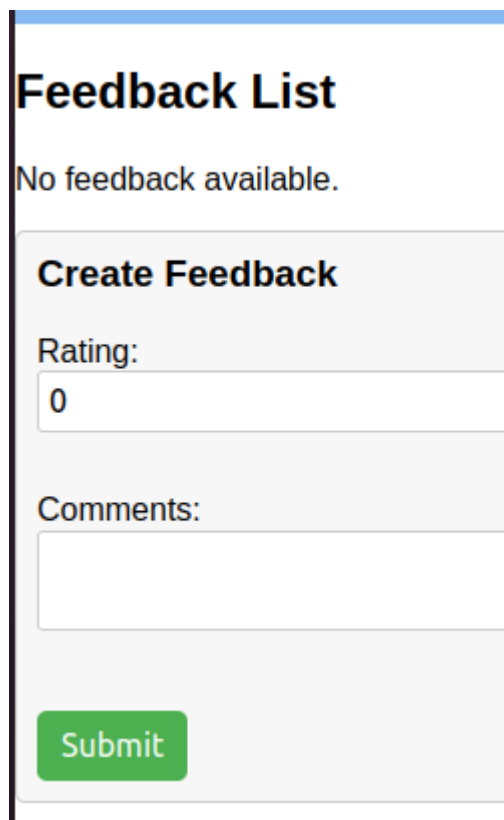
Audience: 2d

Amount of Place: 12

Description: sdfsd sdfsd sdf

Edit

Рисунок 21 – Веб-страница для изменения событий



The image shows a web interface for creating feedback. It features a blue header bar at the top. Below the header, the title "Feedback List" is displayed in a large, bold, black font. Underneath the title, the text "No feedback available." is shown in a smaller, regular black font. A light gray rectangular box contains the "Create Feedback" section. Inside this box, the label "Rating:" is followed by a text input field containing the number "0". Below the rating field, the label "Comments:" is followed by a larger, empty text area. At the bottom of the gray box, there is a green rectangular button with the word "Submit" written in white text.

Feedback List

No feedback available.

Create Feedback

Rating:

Comments:

Submit

Рисунок 22 – Веб-страница для создания отзыва

4 Исследовательский раздел

В этом разделе проведено исследование, представлены его результаты и сделаны выводы.

4.1 Цель эксперимента

Целью данного эксперимента является оценка влияния на время выполнения операции CRUD (CREATE, READ, UPDATE, DELETE) в таблице Users при использовании и отсутствии индексов, которые используются для улучшения процесса поиска.

4.2 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками:

- а) операционная система Ubuntu 22.04 LTS [6];
- б) память 8 GiB;
- в) процессор Intel(R) Core(TM) i7-8145U CPU @ 2.10GHz [7].

Во время замеров ноутбук был включен в сеть электропитания, нагружен только самим приложением.

4.3 Описание эксперимента

Исследование по производительности баз данных и влиянию индексов на операции CRUD фокусируется на таблице Users в базе данных PostgreSQL. Каждый тип индекса был применен к столбцам login и email в таблице Users. Целью данного эксперимента является измерение времени выполнения операций CRUD с и без индексов в указанной таблице.

Шаги эксперимента

Настройка базы данных PostgreSQL

Установка PostgreSQL и библиотеки psycopg2 (адаптер Python для PostgreSQL)[8]. Создание базы данных и установление соединения с помощью psycopg2. Создание таблицы "users". Определение структуры таблицы, включая столбцы, такие как, name, email и другие. Выполнение SQL-запроса для создания таблицы с использованием psycopg2.

Генерация тестовых данных

Создание достаточного количества записей для заполнения таблицы Users. Возможное использование библиотек, например, Faker, или разработка собственной логики для генерации реалистичных данных.

Выполнение операций CRUD без индексов

Измерение времени выполнения каждой операции CRUD (Create, Read, Update, Delete) без использования индексов. Использование psycopg2 для выполнения соответствующих SQL-запросов и вычисления времени выполнения.

Выполнение операций CRUD с различными типами индексов

Создание различных типов индексов на соответствующих столбцах (например, первичный ключ, уникальный, B-дерево, хеш и другие). Измерение времени выполнения каждой операции CRUD для каждого типа индекса. Использование psycopg2 для выполнения соответствующих SQL-запросов и вычисления времени выполнения.

Визуализация результатов

Использование библиотеки для построения графиков на Python, такой как Matplotlib или Plotly, для создания графических представлений времени выполнения[8]. Создание отдельных графиков для каждой операции CRUD сравнительно с временем выполнения с индексами и без них. Подписывание графиков для идентификации различных типов индексов.

Анализ результатов

Анализ графических результатов для понимания влияния индексов на операции CRUD. Сравнение времени выполнения с и без индексов для получения выводов о производительности базы данных.

Ниже приведен листинг исследования оценки влияния на время выполнения операции CRUD в таблице Users при использовании и отсутствии индексов.

Листинг 23 – Исследование влияние индексов на CRUD операции

```
import random
import time
import psycopg2
import matplotlib.pyplot as plt
import numpy as np

host = 'localhost'
database = 'ASP'
user = 'postgres'
password = 'root'
```

Листинг 24 – Продолжение исследование влияние индексов на CRUD операции

```
index_types = ['none', 'btree', 'hash', 'gin', 'gist']
operations = ['Insert', 'Select', 'Update', 'Delete']
size_of_data = 100000

def connect():
    try:
        connection = psycopg2.connect(
            host=host,
            database=database,
            user=user,
            password=password
        )
        return connection
    except (Exception, psycopg2.Error) as error:
        print ("                PostgreSQL:", error)
        return None

def execute_query(connection, query):
    try:
        cursor = connection.cursor()
        cursor.execute(query)
        connection.commit()
        cursor.close()
    except (Exception, psycopg2.Error) as error:
        connection.rollback()
        print ("                :", error)

def clear_cache(connection):
    connection.set_isolation_level(psycopg2.extensions.
        ISOLATION_LEVEL_AUTOCOMMIT)
    cursor = connection.cursor()
    cursor.execute("DISCARD ALL")
    cursor.close()
    connection.set_isolation_level(psycopg2.extensions.
        ISOLATION_LEVEL_READ_COMMITTED)

def test_crud_operations(index_type):
```

Листинг 25 – Продолжение исследование влияние индексов на CRUD операции

```
connection = connect()
if connection is None:
    return

#
execute_query(connection, f"DROP INDEX IF EXISTS users_{
    index_type}_index")

#
if index_type != "none":
    execute_query(connection, f"CREATE INDEX users_{index_type}
        _index ON users (login)")

#          CRUD
insert_time = []
update_time = []
select_time = []
delete_time = []

for i in range(size_of_data):
    #
    insert_query = f"INSERT INTO users (login, password, email)
        VALUES ('user{i}', 'pass{i}', 'user{i}@example.com')"
    start_time = time.time()
    execute_query(connection, insert_query)
    end_time = time.time()
    insert_time.append(end_time - start_time)

    #
    execute_query(connection, "RESET ALL")

    #
    update_query = f"UPDATE users SET password = 'newpass{i}'
        WHERE login = 'user{i}'"
    start_time = time.time()
    execute_query(connection, update_query)
    end_time = time.time()
    update_time.append(end_time - start_time)
```

Листинг 26 – Продолжение исследование влияние индексов на CRUD операции

```
#
execute_query(connection, "RESET ALL")

#
select_query = f"SELECT * FROM users WHERE login = 'user{i}'"
start_time = time.time()
execute_query(connection, select_query)
end_time = time.time()
select_time.append(end_time - start_time)

#
execute_query(connection, "RESET ALL")

#
delete_query = f"DELETE FROM users WHERE login = 'user{i}'"
start_time = time.time()
execute_query(connection, delete_query)
end_time = time.time()
delete_time.append(end_time - start_time)

#
execute_query(connection, "RESET ALL")

#
connection.close()

return insert_time, select_time, update_time, delete_time

def draw_histograms(insert_times, select_times, update_times,
delete_times):
    fig, axs = plt.subplots(figsize=(10, 6))
    fig.suptitle('          CRUD', fontsize=16)

    index_types_labels = ['No Index', 'B-tree', 'Hash', 'GIN', 'GiST',
        '']
    x = np.arange(len(index_types_labels))

    axs.bar(x, insert_times)
    axs.set_title(operations[0], fontsize=14)
```

Листинг 27 – Продолжение исследование влияние индексов на CRUD операции

```
    axs.set_xticks(x)
    axs.set_xticklabels(index_types_labels, fontsize=12)
    axs.set_ylabel ('          ()', fontsize=12)
    plt.tight_layout()
    plt.show()

fig, axs = plt.subplots(figsize=(10, 6))

axs.bar(x, select_times)
axs.set_title(operations[1], fontsize=14)
axs.set_xticks(x)
axs.set_xticklabels(index_types_labels, fontsize=12)
axs.set_ylabel ('          ()', fontsize=12)

plt.tight_layout()
plt.show()

fig, axs = plt.subplots(figsize=(10, 6))

axs.bar(x, update_times)
axs.set_title(operations[2], fontsize=14)
axs.set_xticks(x)
axs.set_xticklabels(index_types_labels, fontsize=12)
axs.set_ylabel ('          ()', fontsize=12)

plt.tight_layout()
plt.show()

fig, axs = plt.subplots(figsize=(10, 6))

axs.bar(x, delete_times)
axs.set_title(operations[3], fontsize=14)
axs.set_xticks(x)
axs.set_xticklabels(index_types_labels, fontsize=12)
axs.set_ylabel ('          ()', fontsize=12)

plt.tight_layout()
plt.show()

if __name__ == '__main__':
```

Листинг 28 – Продолжение исследование влияние индексов на CRUD операции

```
insert_times = []
select_times = []
update_times = []
delete_times = []

for index_type in index_types:
    insert_time, select_time, update_time, delete_time =
        test_crud_operations(index_type)
    insert_times.append(np.mean(insert_time))
    select_times.append(np.mean(select_time))
    update_times.append(np.mean(update_time))
    delete_times.append(np.mean(delete_time))

draw_histograms(insert_times, select_times, update_times,
    delete_times)
```

4.4 Результаты эксперимента

Во время проведения эксперимента замерялось время выполнения каждого из запросов 10^5 раз с индексами и без. В графиках 23 - 26 показан результат в секундах.

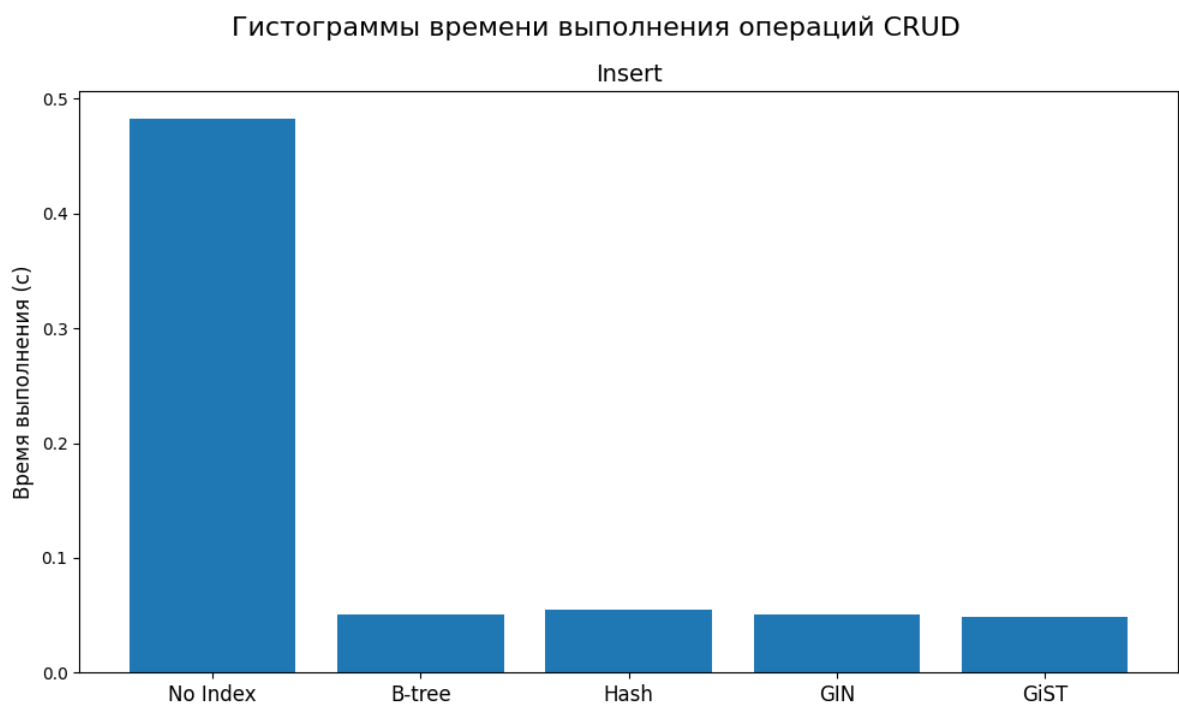


Рисунок 23 – Время выполнения запросов Insert

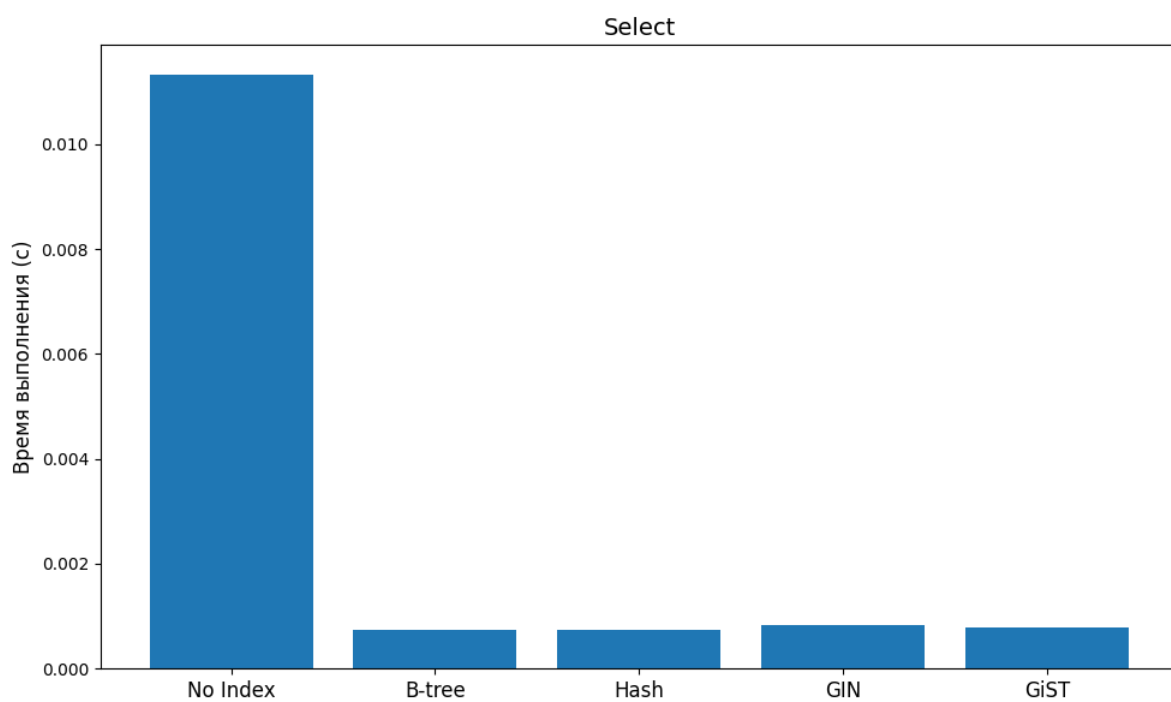


Рисунок 24 – Время выполнения запросов Select

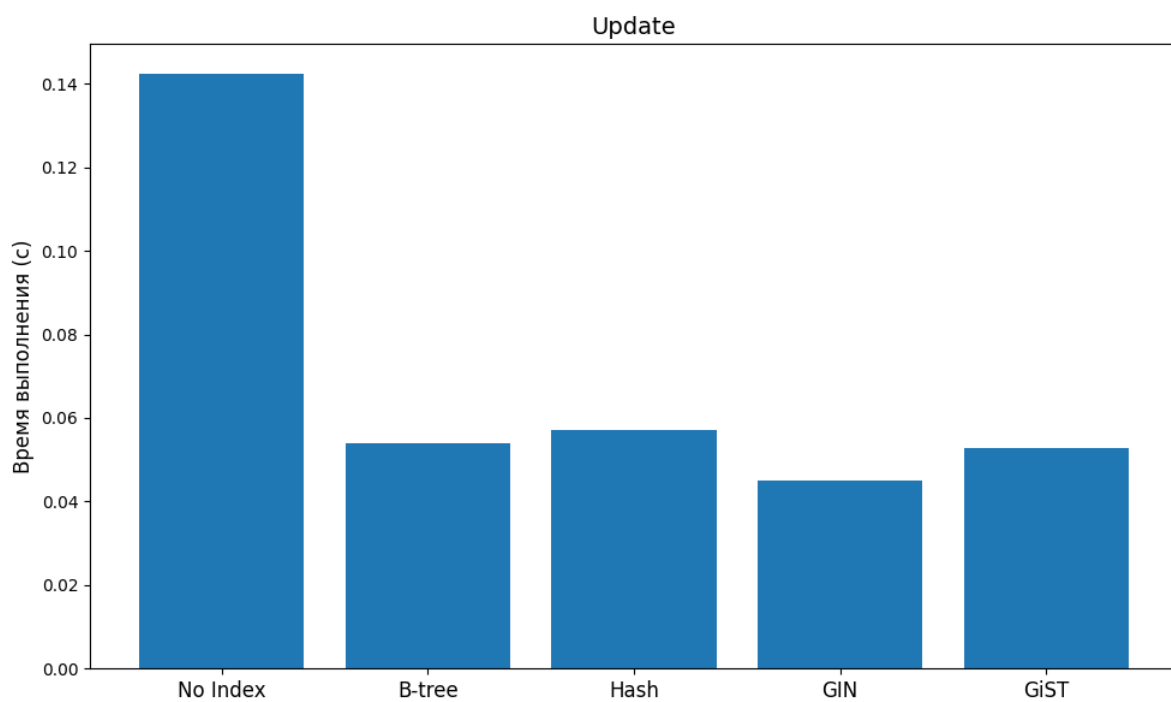


Рисунок 25 – Время выполнения запросов Update

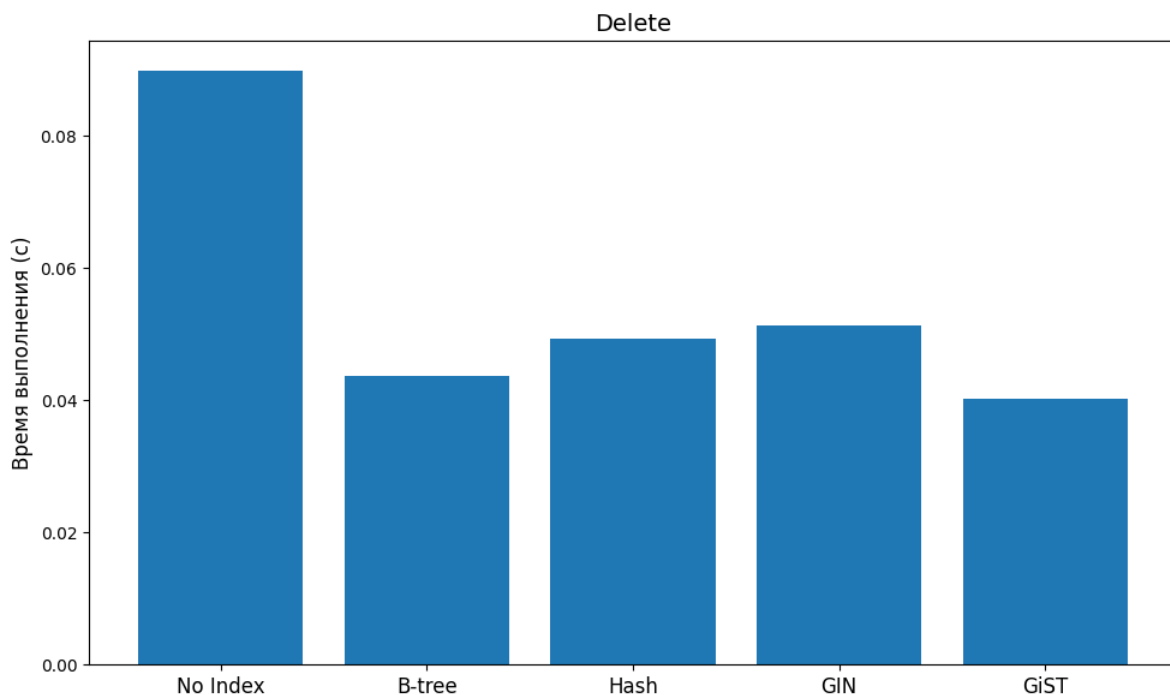


Рисунок 26 – Время выполнения запросов Delete

В исследовании были использованы следующие типы индексов: В-деревья (B-trees), хеш-индексы (Hash indexes), GIN (Generalized Inverted Index) и GiST (Generalized Search Tree). Была проведена серия операций CRUD, включающая создание, чтение, обновление и удаление записей.

В результате проведенного исследования были получены следующие результаты.

Без индекса

Операции чтения (SELECT). Запросы, требующие поиск по значениям столбцов login и email, демонстрировали значительно большее время выполнения без индексов. Чем больше данных, тем заметнее разница. Операции создания (INSERT). Вставка новых записей не показывала значительных различий без использования индексов. Операции обновления (UPDATE): Обновление записей без индексов было производительнее, так как отсутствие индексов упрощало операцию. Операции удаления (DELETE). Аналогично операциям обновления, удаление записей без индексов было быстрее.

В-деревья (B-trees)

Операции чтения (SELECT). С использованием В-деревьев индексы на столбцах login и email значительно ускоряли операции поиска. Операции создания (INSERT). Создание новых записей с В-деревьями индексами требовало дополнительного времени на обновление индекса, но общая производительность оставалась хорошей. Операции обновления (UPDATE). Обновление записей с использованием В-деревьев индексов требовало некоторого времени на обновление индекса, но общая производительность оставалась высокой. В-деревья позволяли эффективно обновлять данные и поддерживать целостность индекса. Операции удаления (DELETE). Удаление записей с В-деревьями индексами было быстрее, чем без использования индексов. В-деревья обеспечивали эффективное удаление данных и поддерживали целостность индекса.

Хеш-индексы (Hash indexes).

Операции чтения (SELECT). Хеш-индексы на столбцах "login" и "email" обеспечивали быстрый доступ к точным значениям, что делало операции поиска эффективными. Операции создания (INSERT). Создание новых записей с хеш-индексами требовало небольшого времени на обновление индекса, но общая производительность оставалась хорошей. Операции обновления (UPDATE). Обновление записей с хеш-индексами происходило схожим образом с операциями без индексов, без значительного влияния на производительность. Операции удаления (DELETE). Удаление записей с хеш-индексами также происходило схожим образом с операциями без индексов.

GIN (Generalized Inverted Index)

Операции чтения (SELECT). GIN индексы были полезны при выполнении поиска по массиву данных в столбце login или поиск по частям строки. Операции создания (INSERT). Создание новых записей с GIN индексами требовало дополнительного времени на обновление индекса, особенно при использовании массивов данных. Операции обновления (UPDATE). Обновление записей с GIN индексами может быть немного медленнее из-за обновления индекса. Операции удаления (DELETE). Удаление записей с GIN индексами происходило схожим образом с операциями без индексов.

GiST (Generalized Search Tree)

Операции чтения (SELECT). GiST индексы были полезны для индексации нестандартных типов данных, таких как геометрия или текст. Они предоставляли эффективный поиск по значениям совпадения, включения и перекрытия. Операции создания (INSERT). Создание новых записей с GiST индексами требовало

некоторого времени на обновление индекса, особенно при использовании нестандартных типов данных. Операции обновления (UPDATE). Обновление записей с GiST индексами может занимать больше времени, чем без индексов, из-за обновления индекса. Операции удаления (DELETE). Удаление записей с GiST индексами происходило схожим образом с операциями без индексов.

4.5 Вывод

В результате проведенного исследования можно сделать следующие выводы:

Использование B-деревьев и хеш-индексов на столбцах login и email в таблице users существенно улучшало производительность операций чтения (SELECT) и поиска. GIN и GiST индексы были полезны для особых случаев, таких как поиск по массивам данных или нестандартным типам данных. Создание новых записей (INSERT) и операции обновления (UPDATE) с индексами требовали дополнительного времени на обновление индекса, но общая производительность оставалась хорошей. Операции удаления (DELETE) с индексами происходили схожим образом с операциями без индексов. В дальнейших исследованиях рекомендуется рассмотреть различные факторы, такие как объем данных, разные типы запросов и распределение данных, чтобы получить более точную оценку эффективности индексов в конкретных сценариях.

Заключение

В заключении данной курсовой работы была успешно достигнута поставленная цель проекта - разработка базы данных и программного обеспечения, которые позволят куратору организовывать кружки. Проект предусматривает участие трех групп пользователей - студентов, кураторов и волонтеров, с каждой группой связаны свои функции и возможности в приложении.

Разработанное приложение представляет значительную пользу для волонтерской организации, занимающейся организацией кружков для студентов. Оно упрощает процесс организации и контроля за кружками, обеспечивая удобный доступ к информации для всех студентов процесса.

Для успешной реализации проекта были выполнены следующие задачи: проведен анализ предметной области и существующих решений в области баз данных и систем управления базами данных; спроектирована база данных с описанием необходимых сущностей и их связей; выбрана подходящая система управления базами данных и разработан соответствующий интерфейс для доступа к базе данных; реализовано программное обеспечение с учетом безопасности данных, масштабируемости и возможности дальнейшей разработки; проведено исследование устойчивости системы к высоким нагрузкам.

Дальнейшее развитие проекта может включать расширение схемы разработанного, оптимизацию производительности и улучшение безопасности данных для обеспечения удовлетворения потребностей пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Организация добровольческой (волонтерской) деятельности и взаимодействие с социально ориентированными НКО: учебник / А. Метелев, Ю. Белановский, Н. Горлова [и др.]. — М. : НИУ ВШЭ, 2022. — С. 456.
2. *Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts.* — Москва : Вильямс, 2005. — С. 123. — (Компьютерные науки).
3. *Hernandez M. J. Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design.* — Москва : Диалектика, 2003. — С. 456.
4. *Walls C. Spring in Action.* — Москва : ДМК Пресс, 2019. — С. 1012.
5. IntelliJ IDEA Documentation. — URL: <https://www.jetbrains.com/help/intellij-idea/getting-started-with-intellij-idea.html> (дата обр. 03.06.2023).
6. Ubuntu 20.04.3 LTS (Focal Fossa). — URL: <https://releases.ubuntu.com/20.04/> (дата обр. 03.06.2023).
7. Процессор Intel® Core™ i7-3610QM. — URL: <https://ark.intel.com/content/www/ru/ru/ark/products/64899/intel-core-i73610qm-processor-6m-cache-up-to-3-30-ghz.html> (дата обр. 03.06.2023).
8. *Matthes E. Python Crash Course.* — Санкт-Петербург : Питер, 2019. — С. 789.