Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Модели данных и системы управления базами данных»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
ассистент кафедры информатики
_____А.В.Давыдчик
___.____.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему
ПРОГРАММНОЕ СРЕДСТВО, РЕАЛИЗУЮЩЕЕ ЛИЧНЫЙ КАБИНЕТ СТУДЕНТА
БГУИР КП 1-40 04 01 028 ПЗ

Выполнил студент группы 153501
Савончик Егор Валерьевич

_____
(подпись студента)
Курсовой проект представлен на проверку ___.____.2024

_____
(подпись студента)

Минск 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Современные образовательные учреждения активно внедряют цифровые технологии для повышения качества обучения и оптимизации административных процессов. В этой связи личные кабинеты студентов становятся важным инструментом взаимодействия между учащимися и университетами. Такие системы предоставляют возможность организовать доступ к образовательным ресурсам, управлять учебной деятельностью, а также получать актуальную информацию о ходе обучения.

С развитием технологий и увеличением объема данных, связанных с учебным процессом, возрастает необходимость создания удобных и функциональных программных средств для управления этой информацией. Личные кабинеты студентов позволяют централизованно хранить данные об успеваемости, расписаниях занятий, учебных материалах и другую важную информацию. Они также обеспечивают прямую связь между студентами и преподавателями, способствуя улучшению коммуникации.

Тем не менее, несмотря на наличие различных платформ и решений, многие из них не полностью удовлетворяют потребности пользователей. Проблемы могут включать в себя сложность интерфейса, недостаточную интеграцию с внутренними системами университета или ограниченные функциональные возможности. Это создает потребность в разработке нового программного обеспечения, которое будет учитывать специфику учебного процесса и современные требования пользователей.

Целью данной курсовой работы является разработка концепции и прототипа программного средства, реализующего функциональность личного кабинета студента. Предполагается, что данное решение позволит эффективно управлять данными, связанными с учебной деятельностью, а также обеспечит интуитивно понятный интерфейс и высокую производительность.

Для достижения поставленной цели необходимо решить следующие задачи:

– провести анализ существующих решений и выявить их сильные и слабые стороны;

– определить основные функции и требования к разрабатываемому программному обеспечению;

– спроектировать структуру базы данных, обеспечивающую надежное хранение и доступ к данным;

– реализовать прототип системы, включающий основные функции личного кабинета студента.

1 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

1.1 Анализ предметной области

Личный кабинет студента представляет собой цифровое решение, предназначенное для предоставления студентам доступа к информации, связанной с их образовательным процессом, и автоматизации взаимодействия с университетом. Это программное средство объединяет данные о расписании занятий, результатах экзаменов, учебных материалах, а также позволяет решать административные задачи, связанные с обучением.

Личный кабинет студента собирает и обрабатывает данные из различных источников, включая внутренние системы университета, такие как системы управления расписанием, базы данных о результатах обучения и системы связи с преподавателями. Это может быть интеграция с электронными журналами, системами управления обучением, а также инструментами коммуникации, такими как электронная почта и мессенджеры.

Возможности личного кабинета:

1 Академические данные. Личный кабинет предоставляет доступ к информации о курсах, результатах промежуточного контроля, итоговых оценках и комментариям преподавателей. Это помогает студентам отслеживать свою успеваемость и планировать дальнейшую учебу.

2 Управление расписанием. Студенты могут просматривать актуальное расписание лекций, семинаров, экзаменов и мероприятий. Некоторые системы позволяют автоматически уведомлять о возможных изменениях, что минимизирует риск пропуска занятий.

3 Учебные материалы. В рамках личного кабинета студента можно предоставлять доступ к лекциям, презентациям, лабораторным заданиям и другим учебным материалам. Это способствует более гибкому и удобному подходу к процессу обучения.

4 Административные функции. Личный кабинет помогает решать задачи, связанные с подачей заявлений, например, на пересдачу экзамена или академический отпуск. Кроме того, в нем можно обновлять личные данные студента, отслеживать задолженности по оплате обучения и получать информацию о состоянии заявлений.

Использование личного кабинета предоставляет студентам удобный доступ к информации, объединяя ее в единой платформе. Это решение упрощает взаимодействие с университетом, позволяет автоматизировать

рутинные процессы и делает образовательный процесс более прозрачным. Автоматическое обновление данных, интеграция с учебными системами и наличие инструментов для анализа успеваемости способствуют модернизации подходов к обучению и повышают эффективность взаимодействия студентов с образовательной средой.

Личный кабинет студента представляет собой удобный инструмент, который позволяет значительно упростить и оптимизировать многие аспекты учебного процесса. Он дает студентам возможность более эффективно управлять своим временем, получать актуальную информацию и своевременно реагировать на изменения в образовательном процессе.

Одним из главных преимуществ является централизованный доступ к информации. В одном месте студент может узнать все важные данные: расписание занятий, результаты экзаменов, текущую успеваемость и уведомления о предстоящих мероприятиях. Это существенно сокращает время, которое студент тратит на поиск информации, и минимизирует риск пропуска важных событий, таких как изменения в расписании или сроки подачи документов.

Также личный кабинет позволяет студенту гибко подходить к организации своего учебного времени. Зная заранее о предстоящих экзаменах или других значимых датах, студент может лучше планировать свою учебную нагрузку, заранее готовиться к важным событиям и избегать конфликтов по времени. Это особенно полезно для тех, кто занимается на разных факультетах или в разных группах, поскольку позволяет легко отслеживать все изменения.

Прямой доступ к результатам экзаменов и промежуточных оценок дает студенту четкое представление о своем уровне знаний. В случае недостаточного результата он может немедленно обратиться за консультацией к преподавателю или начать работу над улучшением оценки. Это помогает более осознанно подходить к процессу обучения и своевременно реагировать на проблемы, избегая сюрпризов в конце семестра.

Особое внимание стоит уделить административным функциям, которые предоставляет личный кабинет. Подача заявлений, например, на пересдачу экзамена, оформление академического отпуска или изменение личных данных, становится гораздо более прозрачной и быстрой. Студент может отслеживать состояние своих заявлений и избегать ненужных визитов в учебные отделы, что значительно упрощает взаимодействие с университетом.

Нельзя не отметить и доступность учебных материалов, которые также часто размещаются в личном кабинете. В отличие от традиционных методов, когда студенту нужно было физически присутствовать на занятиях для

получения лекций или дополнительных материалов, теперь вся информация доступна в любое время и из любого места. Это открывает большие возможности для дистанционного и самостоятельного обучения, особенно в условиях гибридного обучения, когда часть занятий проходит в онлайн-формате.

Системы для личных кабинетов студентов могут значительно различаться по функциональности и архитектуре в зависимости от потребностей учебных заведений и технологий, используемых в их разработке. В общем, можно выделить несколько типов таких систем, которые имеют свои особенности и преимущества.

Одной из таких систем являются платформы, ориентированные на простоту и минимализм. Они часто предоставляют только базовые функции, такие как доступ к расписанию, оценкам и учебным материалам. Такие системы могут быть менее сложными в плане настройки и использования, но они ограничены в возможностях для автоматизации и интеграции с другими университетскими системами. Они идеально подходят для небольших учебных заведений или для образовательных программ с ограниченным количеством участников, где нет необходимости в сложной инфраструктуре.

Другой тип включает более комплексные платформы, которые интегрируют широкий спектр функций, охватывающих различные аспекты образовательного процесса. Эти системы могут включать в себя инструменты для подачи заявлений, организации коммуникации с преподавателями и администрацией, а также могут обеспечивать доступ к различным ресурсам – от учебных материалов до специализированных сервисов для ученых. В таких системах часто реализуются дополнительные функции для мониторинга прогресса студентов, такие как аналитика успеваемости и прогнозирование успеха на основе текущих данных.

Кроме того, существует разница в уровне интеграции с внешними и внутренними системами. В некоторых случаях личные кабинеты могут быть автономными, не связанными с другими программными средствами университета. Это означает, что система работает отдельно от других баз данных и сервисов, что может быть как преимуществом, так и недостатком, поскольку требует регулярного обновления и обработки информации вручную. В других случаях, наоборот, системы могут быть полностью интегрированы с такими сервисами, как электронный журнал, платёжные системы, системы дистанционного обучения, что позволяет минимизировать дублирование данных и обеспечивает более высокий уровень автоматизации.

Особенности таких систем также могут зависеть от типа образовательной программы. Например, для дистанционного образования личные кабинеты могут включать дополнительные средства для общения студентов с преподавателями, доступ к онлайн-курсам и видеоматериалам. В традиционных учебных заведениях, напротив, акцент может быть сделан на функциональность, связанную с физическим посещением лекций и семинаров, а также с возможностью мониторинга посещаемости.

В зависимости от платформы, личные кабинеты могут также различаться по удобству интерфейса, наличию мобильных приложений и особенностям использования на разных устройствах. Некоторые системы могут быть ориентированы на работу с компьютерами, в то время как другие разрабатываются с прицелом на мобильные устройства, что позволяет студентам всегда быть в курсе событий и быстро получать актуальную информацию.

Устройство систем личных кабинетов студентов и подходы к их разработке могут сильно различаться в зависимости от целей, которые ставятся перед системой, и архитектуры, на которой она основана. Обычно такие системы разрабатываются с учетом потребностей учебного заведения, масштабов образовательного процесса и уровня технологий, доступных для их реализации.

Одним из основных аспектов является выбор архитектуры системы. Существуют два основных подхода к созданию личных кабинетов: централизованный и децентрализованный. В централизованной системе все данные и функции находятся в едином хранилище, что позволяет легко управлять информацией и синхронизировать все процессы. Все запросы от пользователей обрабатываются через центральный сервер, и каждый студент получает доступ к данным через интерфейс, связанный с этим сервером. Это решение позволяет унифицировать процесс и упростить управление данными, но также может создать нагрузку на серверы и требовать серьезных затрат на инфраструктуру.

Децентрализованный подход подразумевает более распределенную архитектуру, где данные хранятся в нескольких местах, и каждый из компонентов системы взаимодействует с другими через определенные интерфейсы. Это может быть полезно в случае, если университет использует различные платформы и системы для разных видов деятельности (например, для обучения, управления расписанием, учета оценок), и личный кабинет должен интегрировать данные из нескольких таких источников. Такой подход требует более сложной настройки и разработки интерфейсов для

взаимодействия различных частей системы, но может быть более гибким и масштабируемым.

Кроме того, важным аспектом является выбор типа базы данных, используемой для хранения информации. В зависимости от объемов данных и требований к скорости обработки запросов могут использоваться различные базы данных. Для небольших систем с ограниченным количеством данных может быть достаточно реляционных баз данных, таких как MySQL или PostgreSQL, которые позволяют эффективно работать с текстовой и числовой информацией, обеспечивая надежность и целостность данных. В более крупных системах, где требуется обрабатывать огромные объемы данных в реальном времени, могут использоваться NoSQL базы данных, такие как MongoDB или Cassandra, которые обеспечивают более высокую производительность при работе с неструктурированными данными и могут лучше масштабироваться.

Особое внимание стоит уделить безопасности данных. Личный кабинет студента содержит чувствительную информацию, такую как оценки, результаты экзаменов, личные данные и финансовые транзакции. Поэтому системы должны включать различные уровни защиты, такие как шифрование данных, многоканальную аутентификацию и регулярные аудиты безопасности. Использование технологий блокчейн в некоторых случаях позволяет повысить доверие к системе, обеспечивая прозрачность транзакций и сохранность данных без возможности их изменения.

Разработка интерфейсов пользователя также играет ключевую роль в успешной реализации таких систем. Современные подходы ориентированы на создание интуитивно понятных и удобных интерфейсов, которые позволяют пользователям легко находить нужную информацию и выполнять необходимые действия. В этом контексте активно используются принципы адаптивного дизайна, который позволяет системе работать на разных устройствах – от компьютеров до смартфонов и планшетов.

Таким образом, устройство системы личного кабинета студента зависит от множества факторов, включая требуемую функциональность, количество пользователей, требования безопасности и уровень технологий. Разнообразие подходов и архитектур позволяет создать систему, оптимально подходящую для каждого конкретного образовательного учреждения, с учетом его особенностей и потребностей.

1.2 Обзор существующих аналогов

1.2.1 Blackboard: Инструменты для преподавателей и студентов в образовательном процессе

Blackboard – это система для управления обучением (LMS), которая предоставляет как студентам, так и преподавателям широкий спектр инструментов для организации учебного процесса. Хотя Blackboard включает в себя множество функций для более сложных образовательных процессов, её интерфейс можно настроить таким образом, чтобы студентам был доступен только необходимый функционал. Интерфейс системы Blackboard показан на рисунке 1.1.

Для студентов Blackboard предоставляет следующие возможности:

1 Просмотр расписания. Студенты могут легко отслеживать своё расписание занятий, в том числе время и место проведения лекций и семинаров. В случае изменений в расписании, такие как переносы или отмены, платформа сразу уведомляет студентов об этих изменениях.

2 Оценки и успеваемость. Студенты могут отслеживать свои оценки за курсовые работы, тесты и экзамены. Blackboard предоставляет удобную систему для визуализации оценок, а также обратной связи от преподавателей, что позволяет студентам видеть, на каких аспектах им нужно сосредоточиться для улучшения своей успеваемости.

3 Доступ к учебным материалам. Студенты могут скачать лекции, презентации, задания и другие материалы, предоставленные преподавателями. Эти материалы могут быть загружены в различных форматах, что делает систему удобной для использования и адаптивной к различным образовательным подходам.

4 Общение и уведомления. Через платформу можно легко общаться с преподавателями и одногруппниками, задавать вопросы и участвовать в форумах, а также получать уведомления о новых заданиях, результатах экзаменов и других событиях.

Основной особенностью Blackboard является гибкость системы. Несмотря на её обширную функциональность, которая может быть полезна для более крупных образовательных процессов, в простом варианте можно настроить платформу так, чтобы она обеспечивала только базовые и наиболее востребованные функции для студентов, такие как просмотр расписания и оценок.

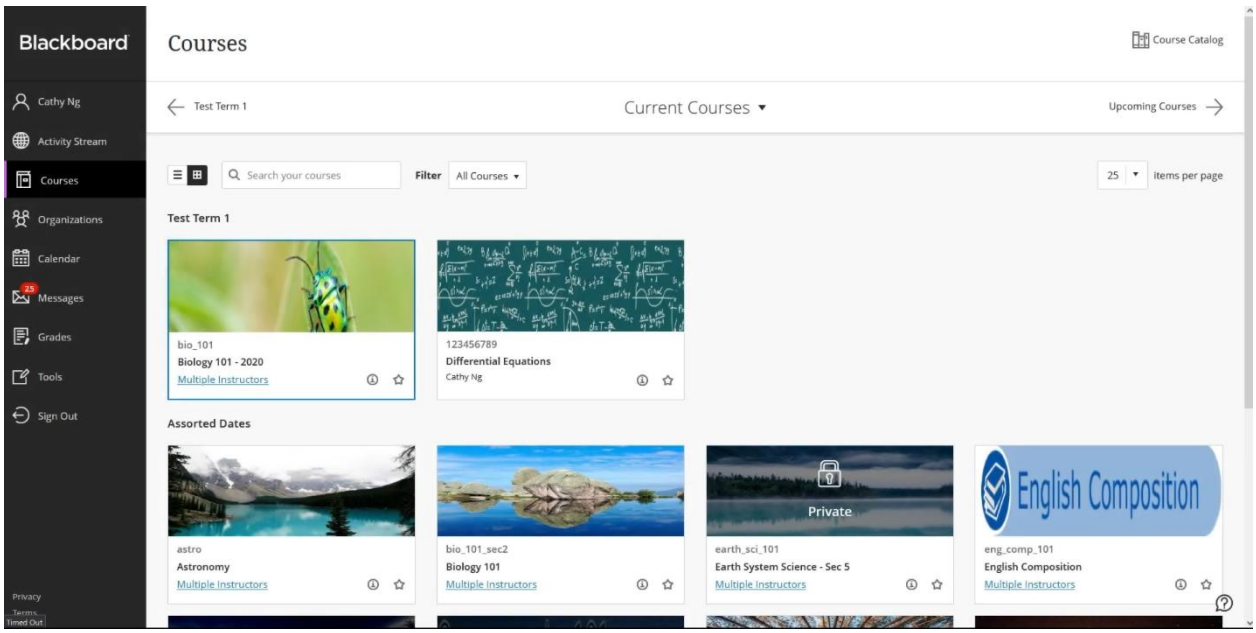


Рисунок 1.1 – Графический интерфейс системы Blackboard

Это комплексная система для управления обучением (LMS), которая используется в образовательных учреждениях для организации учебного процесса. Несмотря на свою функциональность и масштабность, её можно настроить таким образом, чтобы она отвечала лишь базовым требованиям, например, предоставлению личных кабинетов для студентов с доступом к расписанию и оценкам. Устройство Blackboard строится на модульной архитектуре, что позволяет адаптировать систему под разные потребности учебных заведений и конкретных пользователей. [1]

Основной принцип устройства Blackboard заключается в использовании централизованной базы данных, которая хранит всю информацию о курсах, студентам, преподавателях и учебных материалах. Платформа использует реляционные базы данных для обработки информации, что позволяет эффективно управлять большими объемами данных, таких как расписания, оценки, задания и комментарии.

Blackboard состоит из нескольких ключевых компонентов:

1 Серверная часть – это ядро системы, которое обрабатывает все запросы от пользователей (студентов и преподавателей). Серверная часть отвечает за хранение данных, их обработку и взаимодействие с другими компонентами системы. Сервер также управляет доступом и безопасностью данных, а также выполняет функцию синхронизации данных между различными устройствами и пользователями.

2 Интерфейс для пользователей – включает в себя веб-интерфейс, через который студенты и преподаватели получают доступ к системе. Интерфейс довольно интуитивно понятен и имеет множество настраиваемых элементов, что позволяет пользователям быстро находить нужную информацию, будь то расписание, оценки или учебные материалы.

3 Модуль для управления курсами – преподаватели могут создавать, редактировать и публиковать материалы для курса, задавать задания, проводить тесты и следить за успеваемостью студентов. Этот модуль также позволяет загружать различные типы файлов и ресурсы, такие как видео, презентации и документацию.

4 Модуль оценки и успеваемости – студенты могут просматривать свои оценки и результативность по каждому из курсов. Преподаватели, в свою очередь, могут выставлять оценки и оставлять комментарии для студентов, чтобы те могли видеть, в каких аспектах нужно улучшить свою работу.

Что касается классификации Blackboard, то она относится к универсальным платформам для управления обучением с возможностью интеграции с другими образовательными системами и сервисами. Она подходит для широкого спектра образовательных учреждений – от школьных до университетских, и позволяет обучать студентов как в традиционном формате, так и в режиме онлайн. Это означает, что Blackboard может использоваться для управления как очными, так и дистанционными курсами, обеспечивая гибкость в организации образовательного процесса. [1]

Система также поддерживает мобильные версии и приложения для смартфонов, что позволяет студентам получать доступ к своим личным кабинетам и учебным материалам в любое время и в любом месте. Это особенно важно в условиях удаленного обучения, когда доступ к учебным материалам должен быть обеспечен независимо от местоположения учащегося.

1.2.2 Moodle: Платформа для управления обучением и организации учебного процесса

Moodle – это система для управления обучением, которая позволяет образовательным учреждениям создавать платформы для онлайн-обучения, а также для управления учебным процессом в классе. Она широко используется как в крупных университетах, так и в небольших учебных заведениях благодаря своей гибкости и возможности настройки под различные нужды. Moodle имеет открытый исходный код, что позволяет организациям

кастомизировать систему, добавляя или изменяя функционал в зависимости от требований. Интерфейс личного кабинета студента на платформе Moodle показан на рисунке 1.2.

Устройство этой платформы основывается на модульной архитектуре, что позволяет легко добавлять новые функции и компоненты. Центральным элементом системы является сервер базы данных, который хранит всю информацию о курсах, пользователях и учебных материалах. Сервер обрабатывает все запросы пользователей, управляет доступом и защищает данные с помощью современных методов безопасности. [2]

Система состоит из нескольких основных блоков:

1 Панель управления курсами – здесь преподаватели могут создавать новые курсы, добавлять материалы, задания, тесты и форумы для обсуждений. Moodle также позволяет организовать различные виды оценивания: от обычных заданий и тестов до более сложных проектов и семинаров.

2 Интерфейс для студентов – предоставляет доступ к расписанию, учебным материалам, результатам тестов и заданиям. Студенты могут отслеживать свою успеваемость и получать уведомления о новых заданиях и изменениях в курсе. Все эти данные собраны в удобном и структурированном виде, что позволяет пользователю легко ориентироваться в системе.

3 Модуль оценок – это инструмент, через который преподаватели могут выставлять оценки за выполненные задания, тесты или проекты. Студенты могут видеть свои оценки и комментировать их, что позволяет организовать обратную связь между преподавателем и студентом.

4 Мобильное приложение – Moodle также поддерживает мобильные версии, что позволяет студентам и преподавателям получать доступ к учебным материалам и выполнять задания с мобильных устройств. Это делает платформу удобной в использовании и доступной в любом месте, где есть интернет.

В отличие от более крупных и сложных решений для образовательных учреждений, Moodle предлагает оптимальное соотношение между функциональностью и простотой. Система позволяет университетам и колледжам настроить интерфейс и инструменты для решения как базовых, так и более сложных задач, связанных с обучением.

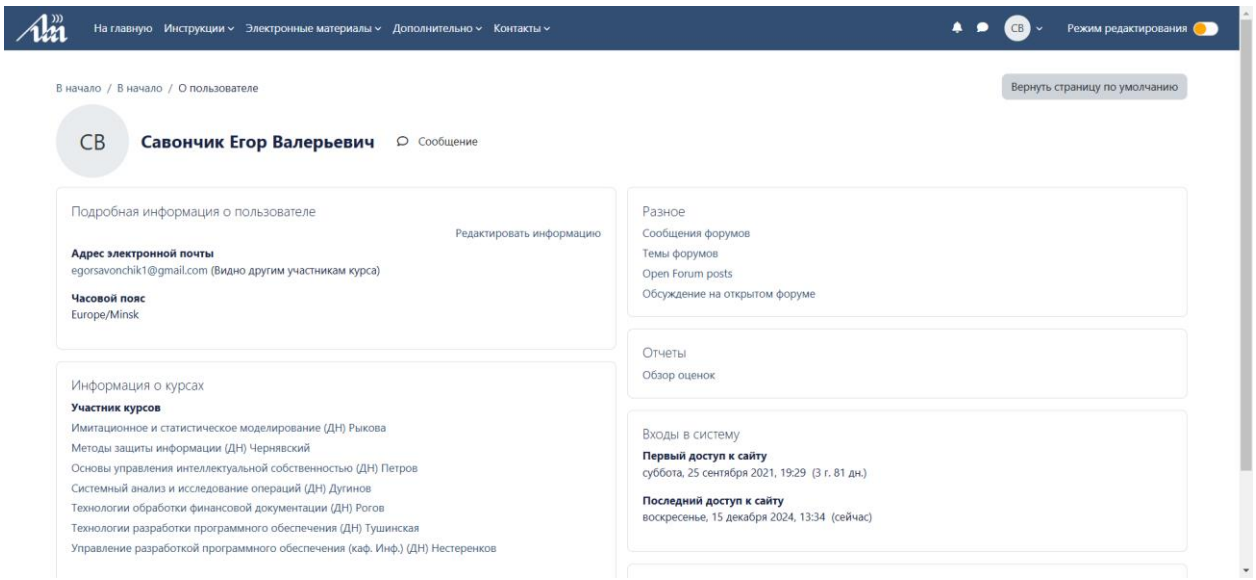


Рисунок 1.2 – Интерфейс личного кабинета студента на платформе Moodle

Внутреннее устройство Moodle не ограничивается только учебным процессом. Платформа также включает механизмы для администрирования, которые позволяют управлять ролями пользователей, правами доступа и настройками безопасности. Это дает возможность настроить систему под конкретные потребности учебного заведения, создавая гибкую структуру, которая будет соответствовать как малым, так и крупным образовательным учреждениям.

Важным элементом системы является её интеграция с внешними инструментами и системами, такими как видеоконференции, инструменты для обработки данных и плагины для различных образовательных целей. Благодаря открытому исходному коду Moodle может быть легко адаптирован для работы с другими платформами и ресурсами, что делает её идеальной для многозадачных и многопрофильных образовательных учреждений. [2]

Таким образом, Moodle представляет собой мощное и многофункциональное решение для управления учебным процессом, которое при этом остаётся достаточно гибким и доступным для образовательных учреждений разного размера и типа.

2 ФОРМИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Функциональные требования к разрабатываемой базе данных

На основе анализа аналогов и определенных задач разрабатываемой системы, выделены функциональные требования для приложения личного кабинета студента. Приложение будет предоставлять доступ к образовательной информации и управлению расписанием, ролями и материалами. Основные функции системы можно разделить на функционал для студентов и суперпользователей (администраторов).

Основные функции для студентов:

1 Авторизация и доступ к личному кабинету.Каждый студент может входить в систему, используя свои учетные данные, чтобы получить доступ к персонализированной информации.

2 Просмотр расписания. Студенты имеют возможность видеть расписание своих занятий, включая дату, время, аудиторию и предмет.

3 Информация о преподавателях. Пользователь может просмотреть данные о преподавателе, который ведет конкретный предмет, включая контактные данные и расписание консультаций.

4 Просмотр информации о курсах. Личный кабинет предоставляет доступ ко всем предметам, на которые студент записан, включая название курса, его описание и связанные материалы.

5 Учебные материалы. Для каждого предмета доступны материалы, загруженные преподавателями: презентации, методички, задания и дополнительные ресурсы.

6 Просмотр оценок. Студенты могут видеть свои оценки за экзамены, контрольные работы и другие формы контроля успеваемости, что помогает им отслеживать свой прогресс.

Основные функции для суперпользователей:

1 Создание учетных записей студентов. Суперпользователь может добавлять новые учетные записи для студентов с указанием основных данных, таких как имя, фамилия, номер зачетной книжки и курс обучения.

2 Управление расписанием. Суперпользователь может составлять и редактировать расписание занятий, включая назначение аудитории, времени и преподавателя для каждого занятия.

3 Добавление преподавателей. В систему могут быть внесены данные о преподавателях, включая их личную информацию, предметы, которые они ведут, и контактные данные.

4 Добавление материалов курса. Суперпользователь может загружать материалы для конкретных курсов или предметов, которые будут доступны студентам через их личный кабинет.

5 Управление учетными записями. Суперпользователь имеет возможность редактировать, удалять или деактивировать учетные записи студентов и преподавателей, а также назначать роли пользователям.

Эти функциональные возможности не только формируют основу для управления образовательным процессом, но и делают систему адаптивной и удобной для всех участников. Для студентов предложенные функции позволяют сосредоточиться на учебе, предоставляя легкий доступ к информации о расписании, учебным материалам и результатам успеваемости. Возможность видеть материалы курса, взаимодействовать с преподавателями через систему и получать обновления делает процесс обучения более прозрачным и управляемым.

Суперпользователи, в свою очередь, получают инструменты для централизованного управления процессом. Функции составления расписания, добавления преподавателей и загрузки материалов позволяют организовать обучение таким образом, чтобы избежать путаницы и повысить эффективность взаимодействия между участниками образовательной среды. Управление учетными записями и ролями дает возможность гибко настраивать систему в зависимости от задач, что особенно важно для крупных образовательных учреждений.

Каждый из предложенных пунктов отвечает конкретным потребностям пользователей и закрывает ключевые задачи, такие как доступ к информации, контроль успеваемости и организация учебного процесса. Внедрение таких возможностей позволяет сократить административные издержки, минимизировать ошибки, связанные с ручным управлением данными, и повысить общий уровень удовлетворенности студентов и преподавателей.

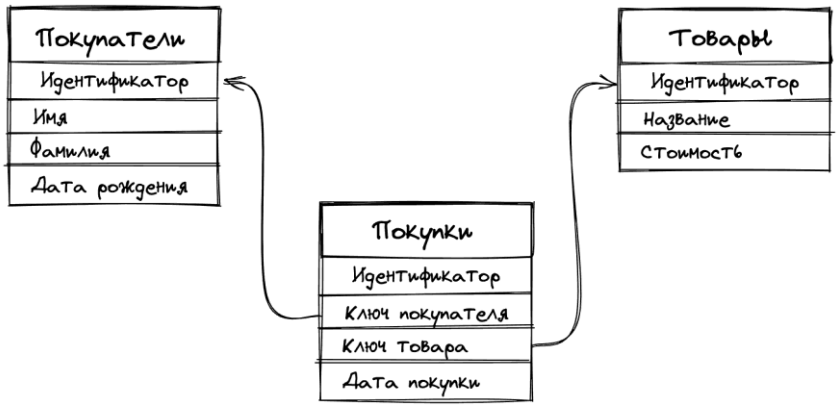
Таким образом, проектируемая система закладывает прочный фундамент для построения современной цифровой образовательной среды, где каждый пользователь получает инструменты, которые делают взаимодействие с университетом удобным, прозрачным и результативным.

2.2 Анализ существующих подходов к разработке баз данных

Проектирование базы данных является важным этапом разработки программного обеспечения, который зависит от особенностей данных, с которыми предстоит работать. Существует несколько подходов к проектированию баз данных, каждый из которых имеет свои сильные стороны

и применяется в зависимости от специфики задачи. Рассмотрим несколько наиболее распространенных подходов.

Реляционная модель – это один из наиболее распространенных подходов в разработке баз данных. В этой модели данные представлены в виде таблиц, где каждая строка таблицы соответствует отдельной записи, а столбцы содержат атрибуты, пример показан на рисунке 2.1. Реляционная модель подходит для приложений, где данные хорошо структурированы и между элементами данных существуют явные связи. Например, для системы личного кабинета студента, где необходимо хранить расписания, оценки, курсы и информацию о преподавателях, реляционная модель будет эффективным решением, так как она позволяет легко управлять связями между такими данными. Использование языка SQL для работы с данными в реляционной базе позволяет формулировать сложные запросы и обеспечивать целостность данных. Такая модель также предоставляет возможность нормализации данных, что способствует уменьшению избыточности и повышению эффективности хранения информации. В реляционных базах данных легко реализовать схемы, обеспечивающие строгую целостность данных, такие как ограничения на уникальность, проверки на целостность связей (например, внешние ключи) и триггеры для автоматизации действий, что является важным аспектом для сохранения точности и безопасности данных в больших системах. Реляционная модель позволяет эффективно использовать индексы для ускорения поиска и обработки запросов, что делает ее подходящей для систем с высокими требованиями к производительности. [3]



Покупатели

Идентификатор	Имя	Фамилия	Дата рождения
1	Илья	Иванов	26-11-1983
2	Сергей	Петров	13-01-2003
—	—	—	—

Рисунок 2.1 – Пример реляционной базы данных

Другим подходом является использование NoSQL баз данных. В отличие от реляционных, NoSQL базы данных не используют таблицы с жесткой схемой. Этот подход включает в себя несколько подкатегорий, показанных на рисунке 2.2, таких как документно-ориентированные базы данных, ключ-значение, графовые базы и другие. Одной из наиболее популярных категорий является документно-ориентированные базы данных, где данные хранятся в виде документов, например, в формате JSON или BSON. Эти базы данных подходят для работы с неструктурированными или слабо структурированными данными, такими как записи с переменным набором полей. NoSQL базы данных обеспечивают высокую производительность и гибкость при масштабировании, что делает их удобными для приложений с высоким объемом данных и гибкой структурой, например, для хранения сообщений пользователей, медиафайлов, комментариев и других элементов, которые не поддаются строгой структуризации. Кроме того, NoSQL базы позволяют эффективно работать с большими объемами данных в реальном времени, что важно для динамических веб-приложений. [4]

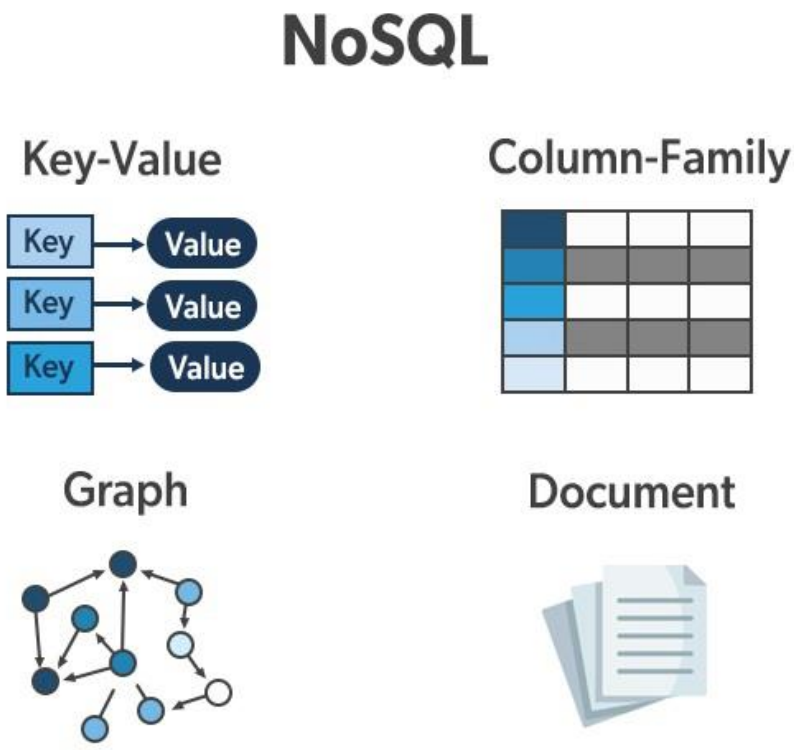


Рисунок 2.2 – Категории NoSql баз данных

В зависимости от специфики задачи, каждое из этих решений имеет свои преимущества и может быть использовано для разных типов систем. Реляционные базы данных подходят для приложений с четко

структурированными данными, в то время как NoSQL базы данных и графовые базы предлагают гибкость и удобство при работе с неструктурированными или взаимосвязанными данными.

Далее рассмотрим более подробно основные отличия реляционных и нереляционных баз данных.

2.2.1 Реляционные базы данных

Реляционные базы данных (РБД) представляют собой модель, в которой данные организованы в виде таблиц (или отношений), состоящих из строк и столбцов. Каждая таблица в РБД описывает один объект (например, студента, курс или расписание), а строки таблицы – это записи, представляющие отдельные экземпляры этих объектов. Столбцы же содержат атрибуты, которые описывают свойства объектов.

Основное преимущество реляционных баз данных заключается в том, что они обеспечивают высокую степень структурированности и позволяют эффективно управлять данными с явными взаимосвязями. Например, в системе личного кабинета студента реляционная модель может использовать таблицы для хранения данных о студентах, курсах, преподавателях, оценках и расписаниях. Каждая из этих таблиц будет иметь определенную структуру, например, таблица студентов может содержать такие поля, как «ID студента», «ФИО», «Дата рождения», а таблица оценок – поля «ID студента», «ID курса», «Оценка», «Дата экзамена».

Связи между таблицами в реляционных базах данных реализуются через ключи. Каждая таблица обычно имеет уникальный идентификатор (первичный ключ), который однозначно идентифицирует запись в таблице. Для установления связи между таблицами используется внешний ключ – это ссылка на первичный ключ другой таблицы. Например, таблица «Оценки» может содержать внешний ключ, который ссылается на первичный ключ таблицы «Студенты», что позволяет отслеживать, какой студент получил какую оценку по конкретному курсу.

Пример структуры реляционной базы данных показано на рисунке 2.3:

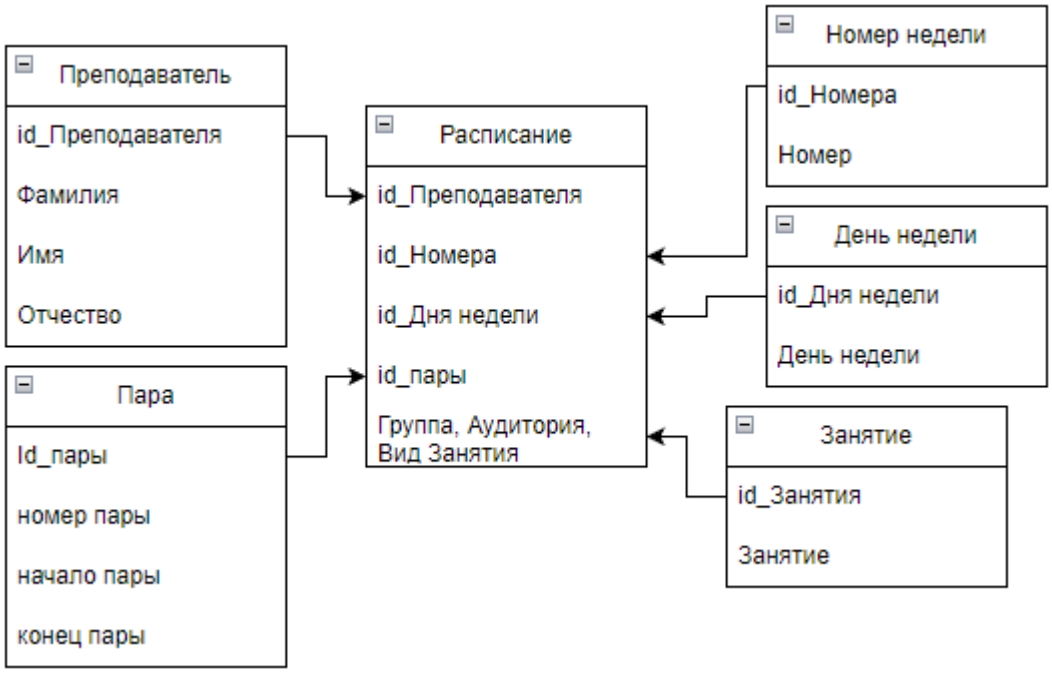


Рисунок 2.3 – Пример структуры реаляционной базы данных

Реляционные базы данных обеспечивают целостность данных с помощью механизмов, таких как ограничения целостности (например, уникальность, проверка значений) и транзакции.

Транзакции в реляционных базах данных обеспечивают важные свойства, известные как ACID (Atomicity, Consistency, Isolation, Durability). Эти свойства критически важны для поддержания целостности и надежности данных, особенно в таких системах, как образовательные платформы, где каждая операция (например, обновление оценок или изменение расписания) должна быть выполнена без ошибок и с гарантией корректности. Рассмотрим каждое свойство более подробно [5]:

1 Атомарность (Atomicity). Это свойство гарантирует, что все операции, входящие в состав транзакции, будут выполнены как единое целое. Если хотя бы одна операция в транзакции не может быть выполнена, вся транзакция откатывается (отменяется), и система возвращается в исходное состояние, как если бы транзакция не выполнялась вовсе. В контексте системы для студентов, например, если транзакция по внесению оценок за экзамен не завершится успешно, то изменения не будут записаны в базу данных, и данные останутся нетронутыми, что предотвращает появление неполных или неверных записей.

2 Согласованность (Consistency). Согласованность гарантирует, что база данных всегда переходит из одного валидного состояния в другое. То есть, если в базе данных есть набор правил и ограничений (например, уникальные

идентификаторы студентов, проверка диапазона оценок), то после выполнения транзакции все эти правила должны оставаться выполненными. Для образовательной системы это означает, что все изменения, связанные с учащимися, курсами, оценками и расписаниями, должны соблюдаться в пределах установленных ограничений и не нарушать структуру базы данных. Например, если студент не может быть зачислен на курс из-за недостаточного количества мест, это правило будет соблюдено благодаря механизму согласованности.

3 Изолированность (Isolation). Это свойство обеспечивает независимость транзакций друг от друга. Если несколько пользователей одновременно пытаются внести изменения в одну и ту же запись базы данных, изолированность гарантирует, что их действия не будут пересекаться или вмешиваться друг в друга, что может привести к ошибкам. Это означает, что изменения одного пользователя (например, обновление оценки) не будут видны другим пользователям, пока транзакция не будет завершена. В образовательной системе это может быть важно, например, когда несколько администраторов или преподавателей одновременно обновляют расписание или оценки – система будет гарантировать, что изменения каждого пользователя выполняются без конфликта.

4 Долговечность (Durability). Долговечность гарантирует, что как только транзакция завершена, ее изменения будут сохранены в базе данных навсегда, даже в случае сбоя системы или отключения питания. Это важно для того, чтобы данные, такие как оценки студентов или изменения в расписании, были надежно зафиксированы и не потерялись после выполнения транзакции. Для образовательных систем это означает, что после того как преподаватель внес изменения в оценки студентов или обновил информацию о курсе, эти изменения будут сохранены в базе данных и не будут утрачены при сбое оборудования или программного обеспечения.

Соблюдение всех этих свойств помогает поддерживать целостность данных в реляционных базах данных, минимизируя вероятность ошибок и обеспечивая надежность работы системы.

При работе с транзакциями в реляционных базах данных важно обеспечивать ACID-свойства. Однако, в многозадачной среде могут возникать определенные проблемы, такие как [5]:

1 Грязное чтение (dirty read) – это ситуация, когда одна транзакция читает данные, которые были изменены другой транзакцией, но не были зафиксированы (commit). Если вторая транзакция отменяет свои изменения (roll back), то первая транзакция будет работать с устаревшими данными, что

может привести к некорректным результатам. Например, студент может видеть оценку, которая еще не была окончательно подтверждена преподавателем.

2 Неповторяющееся чтение (non-repeatable read) – это ситуация, когда данные, прочитанные в одной транзакции, могут быть изменены в другой транзакции, и если тот же запрос повторяется, результаты будут отличаться. В случае с образовательной системой это может привести к тому, что студент увидит разные значения оценок на протяжении одного сеанса.

3 Фантомное чтение (phantom read) – это проблема возникает, когда транзакция читает диапазон данных, но в то время как транзакция выполняется, другие транзакции могут добавить или удалить записи, изменяя результаты предыдущего запроса. Например, при выполнении запроса на список студентов, записанных на курс, другая транзакция может добавить нового студента на тот же курс, что приведет к некорректным данным в исходном запросе.

Чтобы бороться с такими проблемами, реляционные базы данных используют различные уровни изоляции транзакций, каждый из которых решает проблему по-своему.

Уровни изоляции транзакций. Реляционные базы данных предлагают несколько уровней изоляции транзакций, которые позволяют контролировать, какие изменения могут быть видны другим транзакциям [6]:

1 Read Uncommitted (Чтение незафиксированных данных) – самый низкий уровень изоляции. В этом случае могут возникать все типы проблем, включая грязное чтение. Этот уровень не гарантирует ни изоляцию, ни целостность данных.

2 Read Committed (Чтение зафиксированных данных) – предотвращает грязное чтение, так как данные, которые были изменены, но не зафиксированы, не могут быть прочитаны другой транзакцией. Однако проблемы с неповторяющимся чтением все еще могут возникать, поскольку данные могут быть изменены после того, как они были прочитаны.

3 Repeatable Read (Повторяемое чтение) – этот уровень изоляции предотвращает грязное чтение и неповторяющееся чтение. Он гарантирует, что если транзакция несколько раз обращается к одним и тем же данным, то она будет читать их неизменными. Однако фантомные чтения все еще возможны.

4 Serializable (Сериализуемый) – самый высокий уровень изоляции, который гарантирует, что транзакции будут выполняться так, как если бы они шли по очереди, то есть без фантомных чтений. Этот уровень изоляции

предотвращает все возможные проблемы, но может значительно снижать производительность, так как транзакции блокируют доступ к данным друг друга.

Выбор правильного уровня изоляции и использование ACID-свойств позволяют балансировать между производительностью и безопасностью данных, обеспечивая как высокую скорость обработки транзакций, так и их корректность в многозадачных и высоконагруженных системах.

SQL (Structured Query Language) – это язык программирования, предназначенный для работы с реляционными базами данных. Он был разработан для выполнения операций, таких как извлечение, вставка, обновление и удаление данных, а также для управления структурой базы данных. SQL является стандартом в мире реляционных баз данных, и большинство современных СУБД (систем управления базами данных) поддерживают этот язык. Его основные преимущества – это гибкость, мощность и возможность обработки как небольших, так и больших объёмов данных. [7]

В основе SQL лежит несколько принципов, которые обеспечивают его эффективность и универсальность в работе с реляционными базами данных. Прежде всего, SQL позволяет пользователям манипулировать данными, работая с таблицами, которые представляют собой совокупности строк и столбцов. С помощью SQL можно не только изменять данные, но и определять структуру баз данных, создавать новые таблицы, устанавливать связи между ними и обеспечивать целостность информации.

SQL-команды можно разделить на несколько основных категорий, которые показаны на рисунке 2.4, каждая из которых выполняет свою функцию [7]:

1 DML (Data Manipulation Language) – Язык манипуляции с данными. Эти команды отвечают за работу с самими данными в базе данных. С их помощью можно добавлять новые записи, изменять существующие или удалять данные. Команды DML позволяют пользователям манипулировать данными в таблицах, что является важной частью взаимодействия с базой данных.

2 DDL (Data Definition Language) – Язык определения данных. Эти команды позволяют создавать, изменять и удалять объекты базы данных, такие как таблицы, индексы и схемы. DDL-команды работают с самой структурой базы данных.

3 DCL (Data Control Language) – Язык управления доступом. Эти команды позволяют управлять правами пользователей базы данных, что важно для обеспечения безопасности и контроля над доступом к данным.

4 TCL (Transaction Control Language) – Язык управления транзакциями. Эти команды позволяют управлять транзакциями, которые обеспечивают атомарность и целостность данных при выполнении нескольких операций. TCL гарантирует, что изменения в базе данных выполняются правильно и в нужном порядке.

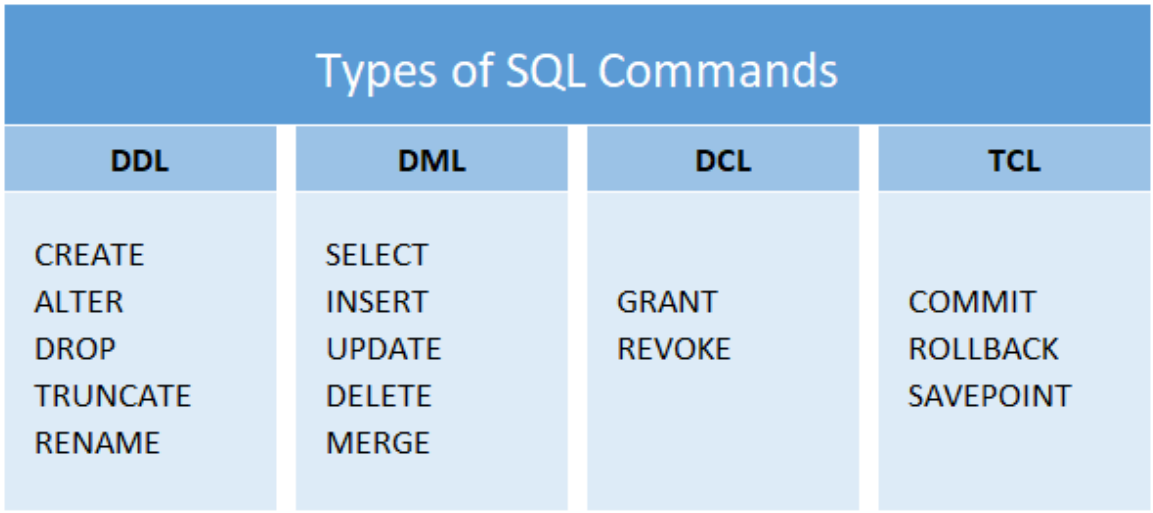


Рисунок 2.4 – Категории комманд Sql

Одна из ключевых особенностей SQL – это возможность создания сложных запросов, которые позволяют извлекать данные из различных таблиц, объединять их и агрегировать. Язык SQL поддерживает различные операции, такие как сортировка, фильтрация и группировка данных, что дает пользователям большую гибкость при работе с базой данных.

Кроме того, SQL включает механизмы для обеспечения целостности данных, такие как ограничения, транзакции и индексы. Например, с помощью ограничений можно задать уникальность данных в столбцах или установить правила для значений (например, проверку на допустимые диапазоны). Транзакции же обеспечивают сохранность данных и их консистентность в случае ошибок или сбоев, гарантируя, что изменения будут выполнены корректно и полностью. Эти возможности делают SQL важным инструментом для работы с критически важной информацией, такой как оценки студентов, расписания, результаты экзаменов и другие данные, где ошибки могут привести к серьёзным последствиям.

Также, реляционные базы данных предлагают возможности для нормализации данных, то есть их организации таким образом, чтобы минимизировать дублирование и избыточность информации. Это помогает повысить производительность системы и улучшить управление данными.

Нормализация данных – это процесс структурирования данных в реляционной базе данных с целью минимизации избыточности и предотвращения аномалий при обновлении, вставке и удалении данных. Этот процесс разделяется на несколько этапов, называемых нормальными формами, каждая из которых уточняет и улучшает структуру данных. В реляционной модели существует несколько нормальных форм, показанных на рисунке 2.5, от первой до пятой, но для большинства приложений достаточно использования первых трех. [8]

Рисунок 2.5 – Перечисление нормальных форм

Первая нормальная форма (1НФ) предполагает, что таблица не содержит повторяющихся групп данных, а каждый атрибут (столбец) должен хранить атомарные (неделимые) значения. Это означает, что в каждой ячейке таблицы может содержаться только одно значение, и никакие сложные структуры, такие как массивы или списки, не могут быть сохранены в одном столбце. Таким образом, таблица в 1НФ будет содержать только простые, атомарные значения для каждого поля.

Вторая нормальная форма (2НФ) строится на основе первой нормальной формы, но требует, чтобы все данные, которые не зависят от первичного ключа, были исключены. Иными словами, она устраняет частичные зависимости, когда атрибуты зависят только от части первичного ключа, а не от всего его состава. В 2НФ каждая неключевая колонка должна зависеть от всего ключа, а не от его части. Эта форма устраняет повторение данных и

улучшает структуру таблиц, особенно если в таблице используются составные ключи.

Третья нормальная форма (3НФ) – это одна из самых широко применяемых нормальных форм, и она предполагает, что все неключевые атрибуты должны зависеть только от первичного ключа, а не от других неключевых атрибутов. Это устраняет транзитивные зависимости, когда один столбец зависит от другого, а тот, в свою очередь, зависит от первичного ключа. В 3НФ каждая неключевая колонка не должна зависеть от других неключевых столбцов, что помогает устранить лишнюю избыточность данных. [8]

Чем выше нормальная форма, тем меньше избыточности и аномалий, но также возрастает сложность структуры базы данных. Третья нормальная форма является оптимальной для большинства приложений, так как она обеспечивает хорошую балансировку между минимизацией избыточности и простотой структуры данных. Однако в некоторых случаях может быть выгодно использование более высоких нормальных форм, например, четвертой или пятой, но для этого часто требуется более сложное разделение данных, что может повлиять на производительность запросов.

Таким образом, нормализация – это процесс, который позволяет повысить производительность системы и улучшить управление данными, особенно в крупных базах данных.

Процесс перехода между нормальными формами является важной частью нормализации базы данных. Он состоит из ряда шагов, которые постепенно устраняют избыточность и аномалии, обеспечивая более эффективное хранение данных. Каждый шаг перехода между формами включает в себя определенные действия по реорганизации данных в таблицах. Рассмотрим, как происходит переход между основными нормальными формами.

Переход от 1НФ к 2Нф. Первоначально, чтобы привести таблицу к первой нормальной форме (1НФ), необходимо убедиться, что все данные являются атомарными. Это значит, что каждый столбец должен содержать только одно значение, и нельзя хранить списки или другие сложные структуры данных в одной ячейке.

После того как таблица приведена в 1НФ, необходимо перейти ко второй нормальной форме (2НФ). Переход заключается в устранении частичных зависимостей, то есть зависимостей, когда атрибуты зависят не от всего первичного ключа, а только от его части. Это обычно происходит в таблицах с составным ключом (когда ключ состоит из нескольких атрибутов).

Шаги перехода:

1 Разделить таблицу на несколько меньших таблиц, чтобы каждый атрибут зависел от всего первичного ключа, а не только от его части.

2 Переместить атрибуты, которые зависят только от части ключа, в отдельные таблицы, создав связи между ними с помощью внешних ключей.

Пример: Если в таблице «Студенты» есть составной ключ, состоящий из «ID студента» и «Код курса», и если в этой таблице также хранится информация о преподавателе, которая зависит только от «Кода курса», то эту информацию о преподавателе следует вынести в отдельную таблицу, связывая ее с таблицей «Студенты» через «Код курса».

Переход от 2НФ к 3НФ. Переход от второй нормальной формы (2НФ) к третьей нормальной форме (3НФ) заключается в устранении транзитивных зависимостей. Это означает, что все атрибуты, не являющиеся частью первичного ключа, должны зависеть непосредственно от первичного ключа, а не через другие атрибуты. [8]

Чтобы привести таблицу к 3НФ, нужно:

1 Проверить, есть ли в таблице атрибуты, которые зависят от других неключевых атрибутов.

2 Если такие зависимости существуют, их нужно устранить, создав новые таблицы, которые будут содержать зависимые атрибуты.

Пример: Если в таблице «Студенты» помимо «ID студента», «Кода курса» и «Оценки» хранится информация о «Факультете», которая зависит от «Кода курса», то для приведения таблицы к 3НФ, данные о факультете должны быть перемещены в отдельную таблицу, так как факультет зависит от курса, а не от студента напрямую.

Процесс нормализации, переходя от одной нормальной формы к другой, помогает улучшить структуру базы данных, устраняя избыточность и аномалии обновления данных. Однако важно помнить, что нормализация требует балансировки между избыточностью и производительностью. В некоторых случаях, например, для улучшения скорости работы системы, могут использоваться денормализованные структуры данных.

2.2.3 Нереляционные базы данных

NoSQL базы данных представляют собой альтернативу реляционным СУБД, предназначенную для обработки больших объемов данных с гибкой или отсутствующей схемой. В отличие от реляционных баз данных, NoSQL системы могут использовать различные модели хранения данных в

зависимости от их структуры и требований. Основные типы NoSQL баз данных включают следующие:

1 Документно-ориентированные базы данных. В этих системах данные хранятся в виде документов, которые представляют собой структуры, часто в формате JSON, BSON или XML. Каждый документ может содержать вложенные данные и различные поля, что позволяет гибко работать с неструктурированными и полуструктурированными данными. Такой подход идеально подходит для хранения информации с переменным набором атрибутов, например, для логов, сообщений, профилей пользователей и прочего. Одним из популярных примеров документно-ориентированной базы данных является MongoDB.

2 Ключ-значение базы данных. Эти базы данных представляют данные в виде пар «ключ-значение». Каждое значение связано с уникальным ключом, что позволяет быстро получать доступ к данным по этому ключу. Такие базы данных отлично подходят для хранения сессий, кеширования, настроек и других данных, где важна высокая скорость обработки запросов. Примеры: Redis, DynamoDB.

3 Колонно-ориентированные базы данных. В этих базах данные организованы по колонкам, а не по строкам. Каждый столбец может быть независимым хранилищем, что делает колонно-ориентированные базы подходящими для аналитических запросов и обработки больших объемов данных. Они часто используются в системах, которые требуют быстрой обработки и анализа данных в реальном времени. Пример: Apache Cassandra, HBase.

4 Графовые базы данных. Графовые базы данных фокусируются на хранении и обработке данных в виде графов, где узлы (вершины) представляют объекты, а рёбра (связи) – отношения между ними. Этот подход полезен для приложений, где важно эффективно анализировать и визуализировать сложные связи, например, в социальных сетях, рекомендационных системах и для анализа сетевых данных. Примеры графовых баз данных: Neo4j, ArangoDB.

Каждый тип NoSQL базы данных обладает своими особенностями и преимуществами, которые определяются спецификой задачи. Например, для задач с интенсивной записью и высокой доступностью можно использовать ключ-значение базы, в то время как для хранения и анализа сложных взаимосвязанных данных оптимальны графовые базы.

Отличие NoSQL баз данных от реляционных систем заключается в подходе к гибкости, масштабируемости и обработке данных. Реляционные

базы данных основываются на строгой структуре и заранее определенной схеме, что делает их идеальными для приложений, где данные четко структурированы, и важна надежная целостность связей между элементами. Этот подход обеспечивает предсказуемость и простоту работы с хорошо организованной информацией, например, в системах учета или финансовых приложениях.

NoSQL базы данных, напротив, ориентированы на работу с большими объемами данных, которые могут быть слабо структурированными, изменчивыми или разнообразными. Их гибкость позволяет быстро адаптироваться к изменениям структуры данных без необходимости перестройки всей системы. [9]

Кроме того, в отличие от реляционной модели, которая масштабируется в первую очередь вертикально (увеличение мощности серверов), NoSQL базы данных изначально спроектированы для горизонтального масштабирования, позволяя добавлять новые серверы для повышения производительности и хранения данных. Это делает их подходящими для распределенных систем и облачных приложений, где требуется обработка данных с высокой нагрузкой и низкой задержкой.

2.3 Выбор инструментов разработки

Для выбора подходящей базы данных для разработки предметной области необходимо учитывать характер данных и специфику их обработки в проекте. Если рассматривать задачу, ориентированную на образовательную систему с личными кабинетами, ключевыми аспектами становятся структура данных, требования к их целостности и необходимость масштабируемости.

Во-первых, данные образовательной системы обладают высокой степенью структурированности. Информация об учащихся, курсах, расписаниях и оценках может быть представлена в виде взаимосвязанных таблиц с четкими атрибутами. Это указывает на необходимость поддержки реляционной модели для эффективного управления данными и обеспечения их целостности.

Во-вторых, требуется учитывать возможность работы с полуструктурированными и неструктурированными данными. Например, преподаватели могут загружать учебные материалы в виде документов или медиафайлов, а студенты – оставлять комментарии или участвовать в форумах. В этом случае важно поддерживать гибкость хранения данных, что

может быть обеспечено интеграцией с NoSQL решениями, такими как документно-ориентированные базы.

Третий важный аспект – производительность и масштабируемость системы. Платформа, работающая с образовательными данными, должна эффективно обрабатывать запросы как от отдельных пользователей, так и от групп. Например, во время регистрации на курсы или проверки расписания в часы пиковой нагрузки требуется высокая скорость доступа к данным. Масштабируемые решения позволят адаптировать систему к увеличению числа пользователей и данных.

Еще один критический фактор – обеспечение транзакционной целостности. Для образовательной системы важно гарантировать правильность данных при их обновлении, например, при изменении расписания или выставлении оценок. Поэтому база данных должна поддерживать механизмы транзакций и уровни изоляции для минимизации ошибок, таких как фантомное чтение или грязное чтение.

Наконец, безопасность и контроль доступа играют ключевую роль. Платформа должна иметь возможность разделять права пользователей (администраторов, преподавателей и студентов) и защищать данные от несанкционированного доступа.

Для реализации поставленных задач важно выбрать систему управления базами данных, которая обеспечит как надежное хранение структурированных данных, так и гибкость в работе с дополнительной информацией, имеющей менее формализованную структуру. В то же время, база данных должна обладать высокой производительностью для обработки запросов от большого числа пользователей, поддерживать транзакционную целостность и предлагать возможности для масштабирования по мере увеличения объема данных и нагрузки.

Рассмотрев возможные решения, можно сделать выбор в пользу системы, которая объединяет преимущества реляционной модели и поддержку современных подходов к работе с данными.

PostgreSQL – это мощная реляционная система управления базами данных с открытым исходным кодом, которая поддерживает SQL и расширенные возможности для работы с данными. Она широко используется благодаря своей надежности, безопасности и гибкости. PostgreSQL сочетает традиционные реляционные возможности с поддержкой современных функций, таких как JSON-формат для хранения полуструктурированных данных и расширенные механизмы индексации. [10]

Причины выбора PostgreSQL для проекта:

1 Поддержка реляционной модели данных. PostgreSQL идеально подходит для работы с четко структурированными данными, такими как расписания, курсы и оценки. Его возможности работы с таблицами позволяют эффективно организовать данные и управлять связями между ними.

2 Расширенная поддержка JSON. Помимо реляционной модели, PostgreSQL поддерживает хранение полуструктурированных данных в формате JSONB. Это особенно полезно для хранения комментариев, описаний материалов или других записей с вариативной структурой.

3 Механизмы транзакционной целостности. PostgreSQL обеспечивает строгую реализацию ACID-свойств транзакций, что критически важно для образовательной системы, где ошибки в данных недопустимы.

4 Широкие возможности масштабируемости. PostgreSQL хорошо работает как в условиях небольших локальных систем, так и в крупных распределенных приложениях, позволяя эффективно справляться с ростом числа пользователей и объема данных.

5 Инструменты контроля доступа. Система управления правами доступа в PostgreSQL позволяет разграничивать действия различных типов пользователей (администраторы, преподаватели, студенты), обеспечивая безопасность данных.

Таким образом, PostgreSQL является оптимальным выбором для образовательной системы. Его функциональные возможности, поддержка как структурированных, так и полуструктурированных данных, а также надежность делают его мощным инструментом для реализации проекта.

3 ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

3.1 Описание связей и сущностей

Стартовой точкой в проектировании базы данных является описание ключевых сущностей, которые взаимодействуют друг с другом и образуют структуру данных. В этой системе выделяются различные сущности, каждая из которых имеет свои атрибуты и играет определенную роль в функционировании базы данных. Связи между этими сущностями формируют логическую структуру данных, обеспечивающую целостность и эффективность работы всей системы. Описание сущностей и их взаимосвязей позволяет создать четкую модель хранения информации и гарантировать правильное взаимодействие данных на всех уровнях системы. В данной базе данных будет рассмотрено 10 основных сущностей, которые являются центральными для ее работы. [11]

Сущности и их атрибуты:

1 Пользователи (Users):

– user_id: Уникальный идентификатор пользователя;
– username: Логин;
– password: Пароль;
– is_superuser: Является ли пользователь суперпользователем.

2 Студенты (Students):

– student_id: Уникальный идентификатор студента;
– group_id: Ссылка на группу, к которой относится студент;
– user_id: Ссылка на учетную запись пользователя;
– first_name: Имя;
– last_name: Фамилия;
– email: Электронная почта;
– phone_number: Номер телефона.

3 Группы студентов (Student Groups):

– group_id: Уникальный идентификатор группы;
– group_name: Название или номер группы.

4 Курсы (Courses):

– course_id: Уникальный идентификатор курса;
– teacher_id: Ссылка на преподавателя;
– course_name: Название курса;
– course_description: Описание курса;
– number_of_hours: Количество часов.

5 Экзамены (Exams):

– exam_id: Уникальный идентификатор экзамена;

– student_id: Ссылка на студента;

– course_id: Ссылка на курс;

– exam_date: Дата экзамена;

– grade: Оценка за экзамен.

6 Преподаватели (Teachers):

– teacher_id: Уникальный идентификатор преподавателя;

– first_name: Имя преподавателя;

– last_name: Фамилия преподавателя;

– email: Электронная почта;

– phone_number: Телефон.

7 Аудитории (Classrooms):

– classroom_id: Уникальный идентификатор аудитории;

– room_number: Номер аудитории;

– building_name: Название корпуса.

8 Расписание (Schedule):

– schedule_id: Уникальный идентификатор записи в расписании;

– course_id: Ссылка на курс;

– classroom_id: Ссылка на аудиторию;

– day_of_week: День недели;

– start_time: Время начала занятия;

– end_time: Время окончания занятия.

9 Регистрация на курсы (Course Registration):

– registration_id: Уникальный идентификатор регистрации;

– student_id: Ссылка на студента;

– course_id: Ссылка на курс;

– registration_date: Дата регистрации на курс.

10 Материалы курса (Course Materials):

– material_id: Уникальный идентификатор материала;

– course_id: Ссылка на курс;

– title: Название материала;

– content: Содержание материала.

Связи между сущностями:

1 Пользователи и Студенты связаны через поле user_id, один пользователь может быть связан с одним или нулем студентов;

2 Студенты и Группы связаны через поле group_id, многие студенты принадлежат одной группе;

3 Студенты и Курсы связаны через таблицу Course Registration, многие студенты могут быть зарегистрированы на многие курсы;

4 Курсы и Преподаватели связаны через поле teacher_id, один курс ведет один преподаватель;

5 Курсы и Экзамены связаны через поле course_id, один курс может быть связан с многими экзаменами;

6 Курсы и Материалы курса связаны через поле course_id, один курс имеет один набор материалов;

7 Аудитории и Расписание связаны через поле classroom_id, одна аудитория может быть связана с многими записями расписания;

8 Курсы и Расписание связаны через поле course_id, один курс может быть связан с многими записями расписания.

Описания сущностей и их связей определяют структуру базы данных и взаимодействие между различными данными. Установленные связи между сущностями помогают организовать данные таким образом, чтобы обеспечить их целостность и удобство обработки. Систематизация этих связей позволяет эффективно управлять информацией о пользователях, курсах, экзаменах и других компонентах платформы, обеспечивая нормальное функционирование базы данных

3.2 Разработка моделей базы данных

Концептуальная модель базы данных представляет собой абстрактное описание структуры данных, которое отображает основные сущности предметной области и взаимосвязи между ними. Она фокусируется на логических аспектах данных, не учитывая специфики их физического хранения или реализации.

Основной задачей концептуальной модели является представление данных в удобной и понятной форме, которая описывает информационные потоки и взаимосвязи в системе. Она служит средством для согласования требований между заказчиком и разработчиком, обеспечивая четкое понимание структуры данных и их назначения.

В отличие от физической модели, концептуальная модель не включает детали, такие как типы данных, индексы или ограничения. Вместо этого она концентрируется на том, какие данные будут храниться и как они связаны друг с другом, что делает её основой для дальнейших этапов проектирования базы данных.

На следующем рисунке представлена концептуальная модель базы данных. Она демонстрирует основные сущности и связи, обеспечивая обзор

структуры данных, которая будет использоваться в разработке системы. Рисунок 3.1 служит отправной точкой для детализации модели на более поздних этапах проектирования.

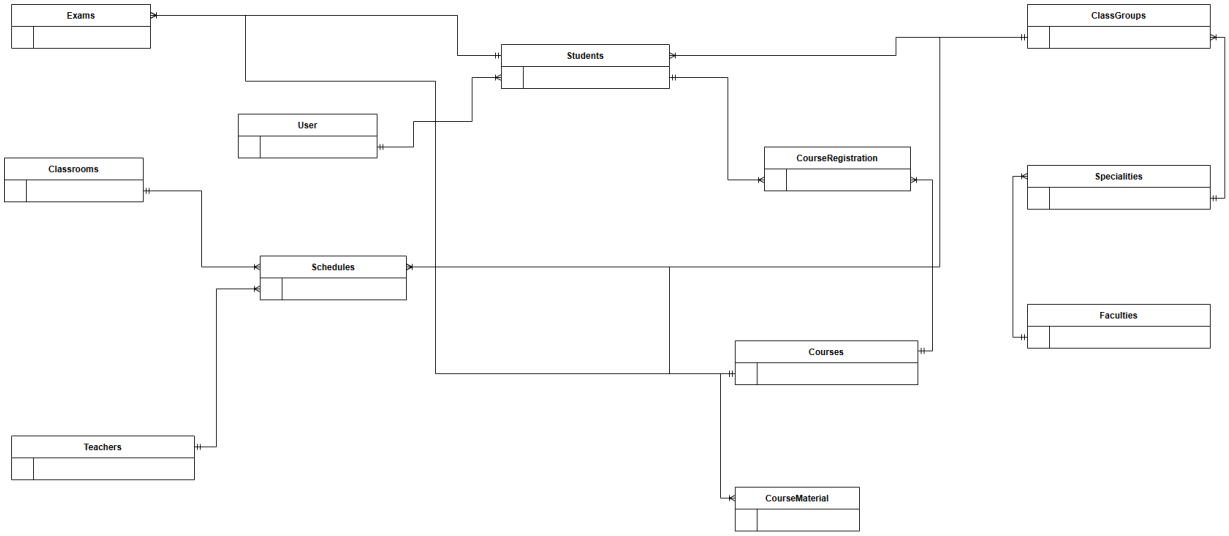


Рисунок 3.1 – Концептуальная модель базы данных

Концептуальная модель базы данных является основой для дальнейшего проектирования, обеспечивая четкое понимание структуры данных и их взаимосвязей на высоком уровне абстракции. Она помогает согласовать видение системы между всеми участниками разработки и задаёт направление для перехода к более детализированным уровням проектирования.

Логическая модель базы данных представляет собой детализацию концептуальной модели, которая включает в себя информацию о типах данных, первичных и внешних ключах, а также других ограничениях, обеспечивающих целостность данных. Логическая модель фокусируется на конкретных требованиях к структуре данных с точки зрения их использования в системе управления базами данных (СУБД), оставаясь при этом независимой от конкретной платформы.

Основной целью логической модели является создание чёткой структуры таблиц, их атрибутов и связей, которая будет эффективно поддерживать операции с данными. Логическая модель включает в себя такие элементы, как нормализация для устранения избыточности данных, определение ограничений для обеспечения корректности информации и подготовку к реализации запросов. [12]

В отличие от концептуальной модели, логическая модель описывает данные более детально, добавляя специфику, которая необходима для

перехода к физической реализации. Она является связующим звеном между абстрактным представлением данных и их конкретным хранением в базе данных.

На следующем рисунке представлена логическая модель базы данных, которая иллюстрирует структуру таблиц, их атрибуты, связи и ограничения. Рисунок 3.2 демонстрирует, как данные организуются для достижения оптимальной производительности и гибкости системы.

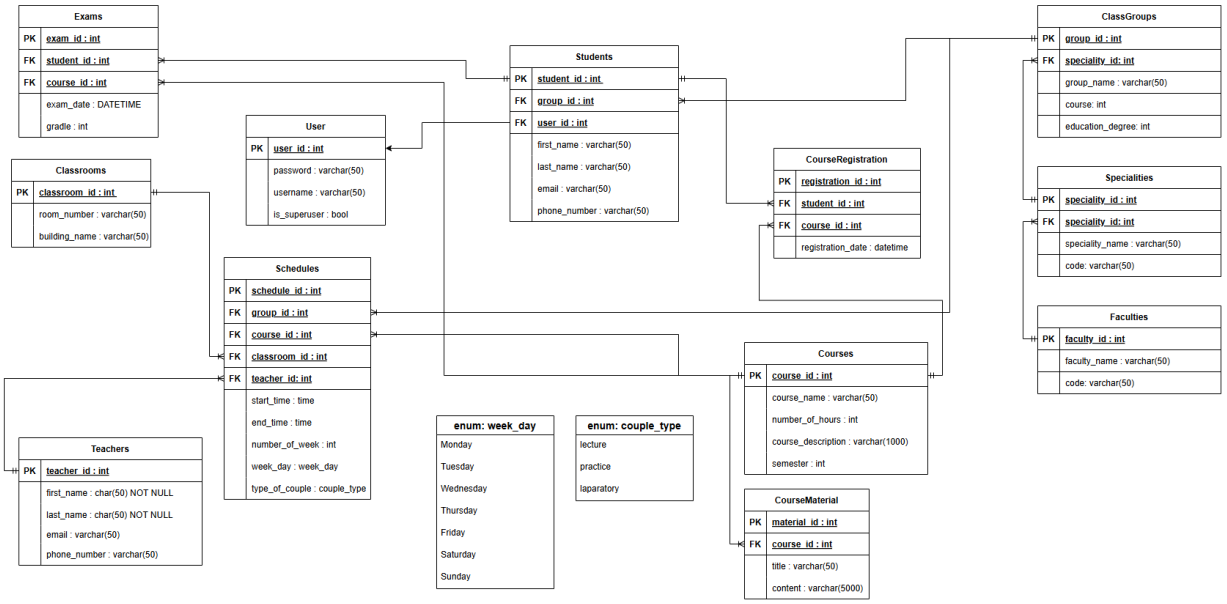


Рисунок 3.2 – Логическая модель базы данных

Логическая модель базы данных обеспечивает точное и детализированное описание структуры данных, которое служит основой для физической реализации. Она помогает адаптировать концептуальную модель к требованиям СУБД, сохраняя логическую целостность и устраняя избыточность данных.

Физическая модель базы данных представляет собой конкретное описание структуры данных с учётом особенностей выбранной системы управления базами данных (СУБД) и аппаратного обеспечения. Она определяет, как данные будут храниться, индексироваться и обрабатываться в реальной среде.

Основная задача физической модели – оптимизация производительности и обеспечение безопасности данных. На этом этапе проектирования учитываются такие аспекты, как выбор типов данных для атрибутов, настройка индексов для ускорения выполнения запросов, создание

таблиц с учетом физического размещения данных, а также реализация механизмов контроля доступа.

Физическая модель тесно связана с логической, но включает в себя реализацию дополнительных технических деталей, таких как способы хранения данных, использование разделов таблиц, настройки транзакций и блокировок. Именно на этапе физической модели проект превращается из абстрактного описания в конкретную базу данных, готовую к эксплуатации.

На рисунке 3.3 представлена физическая модель базы данных, которая демонстрирует, как данные организованы на уровне хранения с учётом параметров производительности и требований системы.

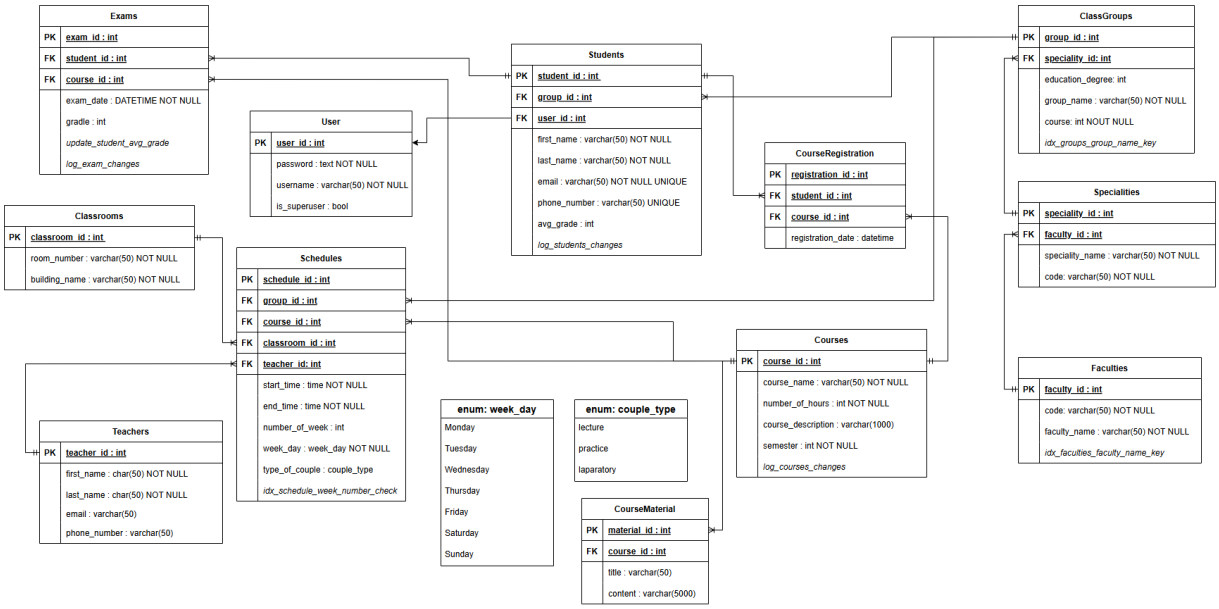


Рисунок 3.3 – Физическая модель базы данных

Конечная модель базы данных представляет собой реализованную версию базы данных, готовую для использования в реальной системе. Она включает в себя все таблицы, индексы, связи и ограничения, а также реализованные хранимые процедуры, триггеры и другие механизмы, обеспечивающие функционирование системы. Конечная модель базы данных представлена в приложении Б.

Процесс проектирования базы данных охватывал все ключевые этапы моделирования: концептуальную, логическую, физическую и конечную модели. Каждая из моделей сыграла свою роль в создании функциональной и эффективной системы управления данными.

4 РАЗРАБОТКА БАЗЫ ДАННЫХ

4.1 Создание исходных таблиц, индексов и ограничений

В дополнение к ускорению выполнения запросов, индексы также помогают улучшить производительность сортировки и фильтрации данных, что делает их незаменимыми для работы с большими объемами информации. Однако важно отметить, что чрезмерное количество индексов на таблицах может привести к излишним затратам на хранение и обновление данных. Поэтому при проектировании базы данных важно сбалансировать количество индексов и их типы в зависимости от предполагаемых операций с данными.

Ограничения (constraints) также играют ключевую роль в обеспечении целостности данных. Они определяют правила для значений в таблицах, например, уникальность значений, отсутствие пустых значений или правильность связей между таблицами. Ограничения могут быть разных типов, включая первичные и внешние ключи, уникальные ограничения и проверки значений. Ограничения могут быть различных типов [13]:

1 PRIMARY KEY – уникальный идентификатор для каждой записи;

2 FOREIGN KEY – связь между таблицами;

3 UNIQUE – гарантирует, что значения в столбце или группе столбцов уникальны;

4 CHECK – проверяет, что значения в столбце удовлетворяют определенному условию;

5 NOT NULL – запрещает вставку пустых значений в столбец.

Кроме того, ограничения позволяют автоматизировать контроль за правильностью данных, минимизируя риски ошибок при вводе или изменении информации. Например, использование внешних ключей гарантирует, что записи в одной таблице всегда соответствуют записям в другой, что особенно важно для поддержания связей между таблицами в сложных моделях. Ограничения на уникальность данных помогают предотвратить дублирование информации, что важно для сохранения точности и надежности базы данных. Также важно отметить, что индексы и ограничения могут значительно улучшить производительность запросов, обеспечивая быстрое выполнение операций на больших объемах данных, если правильно настроены с учетом особенностей нагрузки на систему.

Набор ограничений, используемых в базе данных продемонстрирован на рисунке 4.1.

```
-- Уникальное ограничение для поля group_name в таблице groups
ALTER TABLE groups ADD CONSTRAINT groups_group_name_key UNIQUE (group_name);

-- Уникальное ограничение для поля user_id в таблице students
ALTER TABLE students ADD CONSTRAINT students_user_id_key UNIQUE (user_id);

-- Уникальное ограничение для поля course_name в таблице courses
ALTER TABLE courses ADD CONSTRAINT courses_course_name_key UNIQUE (course_name);

-- Уникальное ограничение для поля course_id в таблице coursematerials
ALTER TABLE coursematerials ADD CONSTRAINT coursematerials_course_id_key UNIQUE (course_id);

-- Уникальное ограничение для поля course_material_id в таблице courses
ALTER TABLE courses ADD CONSTRAINT courses_course_material_id_key UNIQUE (course_material_id);

-- Ограничение для проверки значений поля week_number в таблице schedule (должно быть от 1 до 4)
ALTER TABLE schedule ADD CONSTRAINT schedule_week_number_check CHECK (((week_number >= 1) AND (week_number <= 4)));

-- Уникальное ограничение для поля username в таблице users
ALTER TABLE users ADD CONSTRAINT unique_username UNIQUE (username);

-- Уникальное ограничение для поля email в таблице students
ALTER TABLE students ADD CONSTRAINT unique_email UNIQUE (email);

-- Уникальное ограничение для поля email в таблице teachers
ALTER TABLE teachers ADD CONSTRAINT unique_email_teachers UNIQUE (email);
```

Рисунок 4.1 – Набор ограничений, используемых в проектируемой базе данных

Весь код для создания таблиц, индексов и ограничений будет приведен в Приложении А.

4.2 Создание хранимых процедур

Хранимые процедуры – это заранее подготовленные SQL-операции, которые сохраняются в базе данных и могут быть выполнены по запросу. Они представляют собой наборы SQL-запросов, объединённых логически для выполнения определённой задачи, и часто включают в себя логику обработки данных, условные операторы, циклы и другие элементы. Процедуры значительно упрощают работу с базой данных, так как позволяют централизованно управлять часто повторяющимися операциями, минимизируя количество повторяющегося кода в приложении. [14]

Процедуры активно используются в следующих случаях:

1 Для выполнения сложных вычислений или обработки данных;

2 Для инкапсуляции бизнес-логики;

3 Для обеспечения безопасности и разграничения доступа;

4 Для выполнения регулярных операций по обновлению или обработке данных.

Весь код для создания процедур будет приведен в Приложении А. Ниже представлен список основных процедур, используемых в системе:

1 add_course – процедура служит для добавления нового курса в базу данных.

2 add_student_with_group_id – процедура служит для добавления нового студента в группу по ID.

3 add_student_with_group_name – процедура служит для добавления нового студента в группу по имени группы.

4 add_superuser – процедура служит для добавления пользователя с ролью суперпользователя.

5 add_schedule – процедура служит для добавления расписания для курса и группы.

6 add_schedule_with_names – процедура служит для добавления расписания с использованием имен группы и курса.

7 delete_schedule_by_id – процедура служит для удаления расписания по ID.

8 delete_student_by_id – процедура служит для удаления студента по ID.

9 add_material – процедура служит для добавления учебного материала.

10 add_exam – процедура служит для добавления экзамена в систему.

11 enroll_student – процедура служит для зачисления студента на курс.

12 update_student_grade – процедура служит для обновления оценки студента.

13 update_student_avg_grade – процедура служит для обновления среднего балла студента.

В результате разработки были реализованы все необходимые хранимые процедуры, которые обеспечивают выполнение ключевых операций с данными в базе. Эти процедуры автоматизируют и централизуют обработку информации, улучшая производительность системы и упрощая управление данными.

4.3 Создание функций

Функция в контексте баз данных – это подпрограмма, которая выполняет определенную задачу или вычисление и возвращает результат. Функции в базах данных часто используются для выполнения вычислений, обработки данных или выполнения операций, которые могут быть использованы в SQL-запросах. Функции могут принимать входные параметры и возвращать результат в виде значения или таблицы. [15]

Функции предоставляют важную гибкость в работе с базами данных, позволяя инкапсулировать бизнес-логику, которую можно повторно использовать в различных частях приложения или запросах.

Весь код для создания функций будет приведен в Приложении А. Ниже представлен список основных функций, используемых в системе:

1 get_user_role – функция служит для определения роли пользователя (например, "superuser" или "student") на основе его имени пользователя и пароля.

2 get_grades_by_semester – функция служит для возврата списка оценок студента по курсам за указанный семестр.

3 get_students_in_group – функция служит для возврата списка студентов, которые принадлежат определенной группе, с их основными данными.

4 get_grades_by_username_by_semester – функция служит для возврата списка оценок студента по имени пользователя и семестру.

5 get_student_by_id – функция служит для возврата данных о студенте по его идентификатору.

6 get_schedule_for_group_week – функция служит для возврата расписания для группы на указанную неделю.

7 update_student_avg_grade – функция служит для обновления среднего балла студента после изменения оценок.

В рамках разработки были реализованы все необходимые функции, которые обеспечивают выполнение важных вычислений и операций с данными в базе. Эти функции позволяют централизованно обрабатывать запросы, улучшая производительность и удобство работы с базой данных. Они решают задачи, такие как получение информации о пользователях, оценках, расписаниях и других данных, что делает систему более эффективной и легкой в использовании.

4.4 Создание триггеров

Триггеры в базе данных – это специальные объекты, которые автоматически выполняют определенные действия при наступлении определенных событий, таких как вставка, обновление или удаление данных. Они позволяют автоматически управлять данными, обеспечивая целостность, контроль и выполнение бизнес-логики в ответ на изменения в базе данных. Триггеры могут быть использованы для реализации сложных правил, ведения логов, обновления связанных данных и других задач. В данном разделе

представлены тригеры, которые были реализованы для обеспечения надежности и функциональности системы. [16]

Весь код для создания триггеров будет приведен в Приложении А. Ниже представлен список основных триггеров, используемых в системе:

1 Триггер log_courses_changes служит для логирования изменений (INSERT, UPDATE, DELETE) в таблице courses. Привязан к функции log_courses_changes, которая записывает описание изменений в таблицу Logs.

2 Триггер log_exams_changes служит для логирования изменений (INSERT, UPDATE, DELETE) в таблице exams. Привязан к функции log_exams_changes, которая записывает описание изменений в таблицу Logs.

3 Триггер log_students_changes служит для логирования изменений (INSERT, UPDATE, DELETE) в таблице students. Привязан к функции log_students_changes, которая записывает описание изменений в таблицу Logs.

4 Триггер update_student_avg_grade служит для автоматического обновления среднего балла студента в таблице students при изменении оценок в таблице exams. Привязан к функции update_student_avg_grade, которая пересчитывает средний балл студента на основе данных из таблицы exams.

Таким образом, все необходимые триггеры были успешно реализованы для автоматического обновления данных и логирования изменений в базе данных. Эти триггеры обеспечивают автоматическое вычисление среднего балла для студентов, а также ведение журнала изменений в нескольких таблицах. Благодаря этим триггерам, удалось упростить управление данными и повысить точность их обработки без необходимости вручную отслеживать изменения.

5 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

5.1 Описание разработанного программного средства

Программное средство представляет собой систему авторизации и управления пользовательскими данными, построенную с использованием Tkinter для разработки графического интерфейса и PostgreSQL для хранения информации о пользователях.

При запуске приложения пользователю предлагается выбрать одну из следующих опций на главном экране: авторизация в системе, регистрация нового пользователя или выход из приложения. В зависимости от выбора, интерфейс изменяется, предоставляя доступ к соответствующему функционалу. При успешной авторизации пользователь попадает в основное окно, где могут быть доступны дальнейшие возможности взаимодействия с системой. Экран, на котором осуществляется выбор действия, изображен на рисунке 5.1.

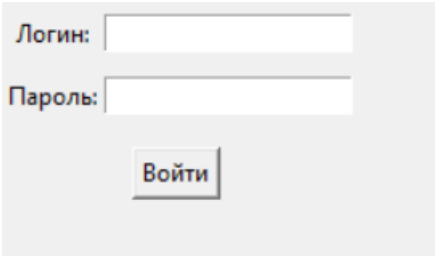


Рисунок 5.1 – Окно, появляющееся при входе в приложение

Интерфейс администратора предоставляет доступ к широкому набору функций для управления данными в системе. После входа в аккаунт администратора открывается главное окно с меню, позволяющим выполнить следующие действия:

1 Добавить занятие – администратор может добавить новое занятие, указав его параметры, включая время, место проведения и связанную группу.

2 Создать аккаунт студента – интерфейс позволяет создать учетную запись студента, вводя основные данные, такие как имя, фамилия, номер группы и контактную информацию.

3 Добавить преподавателя – администратор может зарегистрировать преподавателя, указывая его данные для интеграции в расписание и курсы.

4 Добавить курс – предоставляет возможность создания новых курсов с указанием их описания, количества часов и связанных материалов.

5 Добавить материалы курса – интерфейс позволяет прикрепить учебные материалы, которые могут быть использованы преподавателями и студентами.

6 Назначить курс – позволяет назначить курс определенной группе, связав его с преподавателем и указав временные рамки.

7 Выставить оценку за экзамен – администратор может вручную вводить оценки студентов за экзамены.

8 Создать группу – интерфейс поддерживает создание новых учебных групп с указанием их параметров, таких как название и семестр обучения.

9 Выйти из аккаунта – администратор может завершить работу с системой, вернувшись к окну авторизации.

На рисунке 5.2 продемонстрирован интерфейс приложения для администратора:

Рисунок 5.2 – Интерфейс приложения у администратора

Интерфейс обычного пользователя предоставляет доступ к информации, связанной с учебным процессом. После авторизации студент попадает в главное меню, где может просмотреть свои оценки и расписание занятий. Пользователь получает детализированную информацию о результатах экзаменов по всем предметам, сгруппированным по семестрам, что позволяет легко отслеживать свой учебный прогресс. Также доступен просмотр расписания для выбранной группы на заданную неделю, включая время, преподавателя и место проведения занятий. На рисунке 5.3 продемонстрирован интерфейс приложения для обычного пользователя:

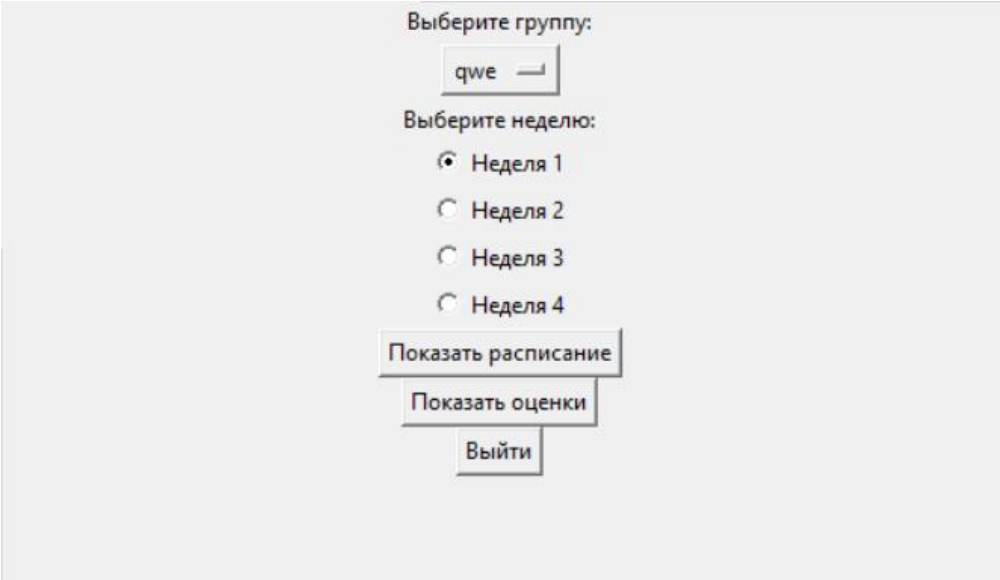


Рисунок 5.3 – Интерфейс приложения у обычного пользователя

Приложение использует библиотеку psycopg2 для подключения и взаимодействия с базой данных PostgreSQL, что обеспечивает надежную и эффективную работу с данными. Для предотвращения SQL-инъекций все SQL-запросы реализованы с использованием параметризации, что значительно повышает уровень безопасности. В текущей версии приложение успешно реализует основные функции и закладывает прочную основу для дальнейшего развития. Возможности расширения включают добавление новых функций, улучшение интерфейса и оптимизацию производительности, что делает приложение перспективным инструментом для управления учебным процессом.

ЗАКЛЮЧЕНИЕ

Результатом работы данного курсового проекта является разработка информационной системы, предназначенной для управления учебным процессом, включая базы данных и пользовательские интерфейсы. Система охватывает основные аспекты работы образовательной организации, такие как учет данных о студентах, курсах, преподавателях, экзаменах и других элементах учебного процесса.

В основе проекта лежит реляционная база данных, реализованная на платформе PostgreSQL, которая обеспечила централизованное хранение данных, надежность и возможности для масштабируемости. СУБД поддерживает использование хранимых процедур, функций и триггеров, которые были созданы для автоматизации задач, таких как обновление средней оценки студента, логирование изменений в таблицах и другие операции. Это позволило минимизировать ручной труд и повысить точность обработки данных.

В процессе работы были созданы интерфейсы для двух типов пользователей: администратора и студента. Администратор получил доступ к функционалу управления учебным процессом, включая добавление курсов, создание аккаунтов, управление расписанием и выставление оценок. Для студентов реализованы функции просмотра их оценок и расписания. Удобство использования системы было достигнуто за счет лаконичного и интуитивно понятного интерфейса.

Поставленные задачи проекта были выполнены: разработана структура базы данных, реализованы механизмы взаимодействия с пользователями, обеспечена безопасность данных благодаря параметризированным запросам, а также протестирована работа ключевых функций. Система предоставляет надежный инструмент для управления учебными процессами и обладает потенциалом для дальнейшего развития и расширения функциональности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Руководство по функциям приложения Blackboard [Электронный ресурс]. – Режим доступа : https://help.blackboard.com/ru-ru/Blackboard_App/Feature_Guide. – Дата доступа : 15.09.2023.

[2] Moodle documentation [Электронный ресурс]. – Режим доступа : https://docs.moodle.org/405/en/Main_page. – Дата доступа : 15.09.2023.

[3] Реляционные базы данных [Электронный ресурс]. – Режим доступа : https://sql-academy.org/ru/guide/relation-databases. – Дата доступа : 15.09.2023.

[4] Нереляционные данные и базы данных NoSQL [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/ru-ru/azure/architecture/data-guide/big-data/non-relational-data. – Дата доступа : 15.09.2023.

[5] Требования ACID на простом языке [Электронный ресурс]. – Режим доступа : https://habr.com/ru/articles/555920/. – Дата доступа : 15.09.2023.

[6] Уровни изоляции транзакций с примерами на PostgreSQL [Электронный ресурс]. – Режим доступа : https://habr.com/ru/articles/317884/. – Дата доступа : 15.09.2023.

[7] Что такое структурированный язык запросов (SQL)? [Электронный ресурс]. – Режим доступа : https://aws.amazon.com/ru/what-is/sql/. – Дата доступа : 15.09.2023.

[8] Описание основных приемов нормализации базы данных [Электронный ресурс]. – Режим доступа : https://learn.microsoft.com/ru-ru/office/troubleshoot/access/database-normalization-description. – Дата доступа : 15.09.2023.

[9] Сравнение SQL- и NoSQL-баз данных [Электронный ресурс]. – https://habr.com/ru/companies/ruvds/articles/727474/. – Дата доступа : 15.09.2023.

[10] Чем PostgreSQL лучше других SQL баз данных [Электронный ресурс]. – Режим доступа : https://habr.com/ru/articles/282764/. – Дата доступа : 15.09.2023.

[11] Лекция 1. Введение в проектирование баз данных [Электронный ресурс]. – Режим доступа : https://msuniversity.ru/d/2/l/2. – Дата доступа : 15.09.2023.

[12] Представление логической и физической моделей данных [Электронный ресурс]. – Режим доступа : https://aws.amazon.com/ru/compare/the-difference-between-logical-and-physical-data-model/. – Дата доступа : 15.09.2023.

[13] Ограничения в SQL [Электронный ресурс]. – Режим доступа : https://codechick.io/tutorials/sql/sql-constraints. – Дата доступа : 15.09.2023.

[14] Хранимые процедуры [Электронный ресурс]. – Режим доступа : https://studfile.net/preview/1602939/page:4/. – Дата доступа : 15.09.2023.

[15] Функции на языке запросов (SQL) [Электронный ресурс]. – Режим доступа : https://postgrespro.ru/docs/enterprise/16/xfunc-sql. – Дата доступа : 15.09.2023.

[16] PostgreSQL-триггеры: создание, удаление, примеры [Электронный ресурс]. – Режим доступа : https://help.sweb.ru/postgresql-triggery-sozdanie-udalenie-primery_1396.html. – Дата доступа : 15.09.2023.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

```
CREATE TYPE day_of_week_enum AS ENUM (
	'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'
);

CREATE TYPE couple_type AS ENUM (
	'lecture', 'practice', 'laboratory'
);

CREATE TABLE Users (
	user_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	username VARCHAR(50),
	password VARCHAR(50),
	is_superuser BOOLEAN
);

CREATE TABLE Groups (
	group_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	group_name VARCHAR(50) UNIQUE
);

CREATE TABLE Classrooms (
	classroom_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	room_number VARCHAR(50),
	building_name VARCHAR(50)
);

CREATE TABLE Teachers (
	teacher_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	first_name VARCHAR(50),
	last_name VARCHAR(50),
	email VARCHAR(50),
	phone_number VARCHAR(50)
```

```
);

CREATE TABLE Students (
	student_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	group_id INT,
	user_id INT UNIQUE,
	first_name VARCHAR(50),
	last_name VARCHAR(50),
	email VARCHAR(50),
	phone_number VARCHAR(50),
	FOREIGN KEY (group_id) REFERENCES Groups(group_id),
	FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE Courses (
	course_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	teacher_id INT,
	course_name VARCHAR(50) UNIQUE,
	course_description VARCHAR(1000),
	number_of_hours INT,
	FOREIGN KEY (teacher_id) REFERENCES Teachers(teacher_id)
);

CREATE TABLE Exams (
	exam_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	student_id INT,
	course_id INT,
	exam_date timestamp,
	grade INT,
	FOREIGN KEY (student_id) REFERENCES Students(student_id),
	FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

CREATE TABLE Schedule (
	schedule_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
	course_id INT,
	classroom_id INT,
	day_of_week day_of_week_enum,
```

```
    couple_type couple_type,
    start_time TIME,
    end_time TIME,
    week_number INT CHECK (week_number >= 1 AND week_number <= 4),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id),
    FOREIGN KEY (classroom_id) REFERENCES Classrooms(classroom_id)
);

CREATE TABLE Registrations (
    registration_id INT GENERATED ALWAYS AS IDENTITY PRIMARY
KEY,
    student_id INT,
    course_id INT,
    registration_date timestamp DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

CREATE TABLE CourseMaterials (
    material_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
    course_id INT UNIQUE,
    title VARCHAR(50),
    content VARCHAR(5000),
    FOREIGN KEY (course_id) REFERENCES Courses(course_id)
);

ALTER TABLE Courses
    ADD COLUMN course_material_id INT UNIQUE;

ALTER TABLE Courses
    ADD CONSTRAINT fk_courses_course_material_id
        FOREIGN KEY (course_material_id) REFERENCES
CourseMaterials(material_id);

CREATE INDEX idx_schedule_week_number ON Schedule (week_number);

CREATE INDEX idx_schedule_day_of_week ON Schedule (day_of_week);
```

```
CREATE INDEX idx_courses_course_name ON Courses (course_name);

CREATE INDEX idx_groups_group_name ON Groups (group_name);

CREATE INDEX idx_classrooms_room_number ON Classrooms (room_number);

CREATE OR REPLACE PROCEDURE add_student_with_group_id(
    student_username VARCHAR(50),
    student_password VARCHAR(50),
    student_first_name VARCHAR(50),
    student_last_name VARCHAR(50),
    student_email VARCHAR(50),
    student_phone_number VARCHAR(50),
    student_group_id INT
)
AS $$
DECLARE
    new_user_id INT;
BEGIN
    -- Вставка данных в таблицу Users и возврат user_id
    INSERT INTO Users (username, password, is_superuser)
    VALUES (student_username, student_password, FALSE)
    RETURNING user_id INTO new_user_id;

    -- Вставка данных в таблицу Students с использованием полученного user_id
    INSERT INTO Students (group_id, user_id, first_name, last_name, email, phone_number)
    VALUES (student_group_id, new_user_id, student_first_name, student_last_name, student_email, student_phone_number);
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE add_student_with_group_name(
    student_username VARCHAR(50),
    student_password VARCHAR(50),
    student_first_name VARCHAR(50),
    student_last_name VARCHAR(50),
```

```
    student_email VARCHAR(50),
    student_phone_number VARCHAR(50),
    student_group_name VARCHAR(50)
)
AS $$
DECLARE
    user_id INT;
    group_id INT;
BEGIN
    SELECT Groups.group_id INTO group_id FROM Groups WHERE group_name = student_group_name;

    IF group_id IS NULL THEN
        RAISE EXCEPTION 'Группа с именем % не найдена.', student_group_name;
    END IF;

    INSERT INTO Users (username, password, is_superuser)
    VALUES (student_username, student_password, FALSE)
    RETURNING Users.user_id INTO user_id;

    INSERT INTO Students (group_id, user_id, first_name, last_name, email, phone_number)
    VALUES (group_id, user_id, student_first_name, student_last_name, student_email, student_phone_number);
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE add_superuser(
    superuser_username VARCHAR(50),
    superuser_password VARCHAR(50)
)
AS $$
BEGIN
    INSERT INTO Users (username, password, is_superuser)
    VALUES (superuser_username, superuser_password, TRUE);
END;
$$ LANGUAGE plpgsql;
```

```
CREATE OR REPLACE PROCEDURE add_schedule(
	p_course_id INT,
	p_teacher_id INT,
	p_group_id INT,
	p_classroom_id INT,
	p_day_of_week day_of_week_enum,
	p_couple_type couple_type,
	p_start_time TIME,
	p_end_time TIME,
	p_week_number INT
)
AS $$
DECLARE
	existing_classroom_schedule_count INT;
	existing_teacher_schedule_count INT;
	existing_group_schedule_count INT;
BEGIN
	-- Проверка наличия пары в заданное время и классе
	SELECT COUNT(*)
	INTO existing_classroom_schedule_count
	FROM Schedule
	WHERE classroom_id = p_classroom_id
		AND day_of_week = p_day_of_week
		AND week_number = p_week_number
		AND ((start_time <= p_start_time AND end_time >= p_start_time)
			OR (start_time <= p_end_time AND end_time >= p_end_time)
			OR (start_time >= p_start_time AND end_time <=
p_end_time));

	IF existing_classroom_schedule_count > 0 THEN
		RAISE EXCEPTION 'Пара уже запланирована в это время в этой
аудитории';
	END IF;

	-- Проверка наличия пары у преподавателя
	SELECT COUNT(*)
	INTO existing_teacher_schedule_count
```

```
    FROM Schedule s
    WHERE s.teacher_id = p_teacher_id
        AND s.day_of_week = p_day_of_week
        AND s.week_number = p_week_number
        AND ((s.start_time <= p_start_time AND s.end_time >= p_start_time)
            OR (s.start_time <= p_end_time AND s.end_time >= p_end_time)
            OR (s.start_time >= p_start_time AND s.end_time <= p_end_time));

    IF existing_teacher_schedule_count > 0 THEN
        RAISE EXCEPTION 'Пара уже запланирована у преподавателя в это время';
    END IF;

    -- Проверка наличия пары у группы
    SELECT COUNT(*)
    INTO existing_group_schedule_count
    FROM Schedule s
    WHERE s.group_id = p_group_id
        AND s.day_of_week = p_day_of_week
        AND s.week_number = p_week_number
        AND ((s.start_time <= p_start_time AND s.end_time >= p_start_time)
            OR (s.start_time <= p_end_time AND s.end_time >= p_end_time)
            OR (s.start_time >= p_start_time AND s.end_time <= p_end_time));

    IF existing_group_schedule_count > 0 THEN
        RAISE EXCEPTION 'Пара уже запланирована у группы в это время';
    END IF;

    -- Вставка данных в таблицу расписания
    INSERT INTO Schedule (course_id, classroom_id, group_id, teacher_id, day_of_week, couple_type, start_time, end_time, week_number)
    VALUES (p_course_id, p_classroom_id, p_group_id, p_teacher_id, p_day_of_week, p_couple_type, p_start_time, p_end_time, p_week_number);
```

```
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE add_schedule_with_names(
	p_course_name VARCHAR(50),
	p_teacher_first_name VARCHAR(50),
	p_teacher_last_name VARCHAR(50),
	p_group_name VARCHAR(50),
	p_building_name VARCHAR(50),
	p_room_number VARCHAR(50),
	p_day_of_week day_of_week_enum,
	p_couple_type couple_type,
	p_start_time TIME,
	p_end_time TIME,
	p_week_number INT
)
AS $$
DECLARE
	existing_classroom_schedule_count INT;
	existing_teacher_schedule_count INT;
	existing_group_schedule_count INT;
	v_course_id INT;
	v_teacher_id INT;
	v_group_id INT;
	v_classroom_id INT;
BEGIN
	-- Получение идентификатора курса по его названию
	SELECT course_id INTO v_course_id FROM Courses WHERE course_name = p_course_name;
	IF v_course_id IS NULL THEN
		RAISE EXCEPTION 'Курс с названием % не найден', p_course_name;
	END IF;

	-- Получение идентификатора преподавателя по его имени и фамилии
	SELECT teacher_id INTO v_teacher_id FROM Teachers WHERE first_name = p_teacher_first_name AND last_name = p_teacher_last_name;
	IF v_teacher_id IS NULL THEN
```

```
        RAISE EXCEPTION 'Преподаватель с именем % % не найден', p_teacher_first_name, p_teacher_last_name;
    END IF;

    -- Получение идентификатора группы по ее названию
    SELECT group_id INTO v_group_id FROM Groups WHERE group_name = p_group_name;
    IF v_group_id IS NULL THEN
        RAISE EXCEPTION 'Группа с названием % не найдена', p_group_name;
    END IF;

    -- Получение идентификатора аудитории по корпусу и номеру аудитории
    SELECT classroom_id INTO v_classroom_id FROM Classrooms WHERE building_name = p_building_name AND room_number = p_room_number;
    IF v_classroom_id IS NULL THEN
        RAISE EXCEPTION 'Аудитория в корпусе % с номером % не найдена', p_building_name, p_room_number;
    END IF;

    -- Проверка наличия пары в заданное время и аудитории
    SELECT COUNT(*)
    INTO existing_classroom_schedule_count
    FROM Schedule
    WHERE classroom_id = v_classroom_id
        AND day_of_week = p_day_of_week
        AND week_number = p_week_number
        AND ((start_time <= p_start_time AND end_time >= p_start_time)
            OR (start_time <= p_end_time AND end_time >= p_end_time)
            OR (start_time >= p_start_time AND end_time <= p_end_time));

    IF existing_classroom_schedule_count > 0 THEN
        RAISE EXCEPTION 'Пара уже запланирована в это время в этой аудитории';
    END IF;
```

```
    -- Проверка наличия пары у преподавателя
    SELECT COUNT(*)
    INTO existing_teacher_schedule_count
    FROM Schedule s
    WHERE s.teacher_id = v_teacher_id
        AND s.day_of_week = p_day_of_week
        AND s.week_number = p_week_number
        AND ((s.start_time <= p_start_time AND s.end_time >= p_start_time)
            OR (s.start_time <= p_end_time AND end_time >= p_end_time)
            OR (s.start_time >= p_start_time AND s.end_time <=
p_end_time));

    IF existing_teacher_schedule_count > 0 THEN
        RAISE EXCEPTION 'Пара уже запланирована у преподавателя в
это время';
    END IF;

    -- Проверка наличия пары у группы
    SELECT COUNT(*)
    INTO existing_group_schedule_count
    FROM Schedule s
    WHERE s.group_id = v_group_id
        AND s.day_of_week = p_day_of_week
        AND s.week_number = p_week_number
        AND ((s.start_time <= p_start_time AND s.end_time >= p_start_time)
            OR (s.start_time <= p_end_time AND end_time >= p_end_time)
            OR (s.start_time >= p_start_time AND s.end_time <=
p_end_time));

    IF existing_group_schedule_count > 0 THEN
        RAISE EXCEPTION 'Пара уже запланирована у группы в это
время';
    END IF;

    -- Вставка данных в таблицу расписания
    INSERT INTO Schedule (course_id, classroom_id, teacher_id, group_id,
day_of_week, couple_type, start_time, end_time, week_number)
```

```
        VALUES (v_course_id, v_classroom_id, v_teacher_id, v_group_id,
p_day_of_week, p_couple_type, p_start_time, p_end_time, p_week_number);
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE delete_schedule_by_id(
        p_schedule_id INT
)
AS $$
BEGIN
        DELETE FROM Schedule WHERE schedule_id = p_schedule_id;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE PROCEDURE delete_student_by_id(
        p_student_id INT
)
AS $$
DECLARE
        v_user_id INT;
BEGIN
        SELECT user_id INTO v_user_id FROM Students WHERE student_id =
p_student_id;
        DELETE FROM Students WHERE student_id = p_student_id;

        IF v_user_id IS NOT NULL THEN
                DELETE FROM Users WHERE user_id = v_user_id;
        END IF;
END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION get_student_by_id(
        p_student_id INT
) RETURNS TABLE (
        student_id INT,
        group_id INT,
        user_id INT,
        first_name VARCHAR(50),
```

```
    last_name VARCHAR(50),
    email VARCHAR(50),
    phone_number VARCHAR(50)
)
AS $$
BEGIN
    RETURN QUERY
    SELECT *
    FROM Students
    WHERE Students.student_id = p_student_id;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_student_by_id(1);

CREATE OR REPLACE FUNCTION get_students_in_group(
    p_group_name VARCHAR(50)
) RETURNS TABLE (
    student_id INT,
    group_id INT,
    user_id INT,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(50),
    phone_number VARCHAR(50)
)
AS $$
BEGIN
    RETURN QUERY
    SELECT s.*
    FROM Students s
    JOIN Groups g ON s.group_id = g.group_id
    WHERE g.group_name = p_group_name;
END;
$$ LANGUAGE plpgsql;

SELECT * FROM get_students_in_group('GroupA');
```

```
CREATE OR REPLACE FUNCTION get_schedule_for_group_week(
	p_group_name VARCHAR(50),
	p_week_number INT
)
RETURNS TABLE (
	day_of_week day_of_week_enum,
	week_number INT,
	couple_type couple_type,
	start_time TIME,
	end_time TIME,
	course_name VARCHAR(50),
	teacher_first_name VARCHAR(50),
	teacher_last_name VARCHAR(50),
	room_number VARCHAR(50)
)
AS $$
BEGIN
	IF NOT EXISTS (SELECT 1 FROM Groups WHERE group_name = p_group_name) THEN
		RAISE EXCEPTION 'Группа с именем % не существует', p_group_name;
	END IF;

	RETURN QUERY
	SELECT
		Schedule.day_of_week,
		Schedule.week_number,
		Schedule.couple_type,
		Schedule.start_time,
		Schedule.end_time,
		Courses.course_name,
		Teachers.first_name as teacher_first_name,
		Teachers.last_name as teacher_last_name,
		Classrooms.room_number
	FROM Schedule
		JOIN Courses ON Schedule.course_id = Courses.course_id
		JOIN Teachers ON Schedule.teacher_id = Teachers.teacher_id
		JOIN Groups ON Schedule.group_id = Groups.group_id
```

```
        JOIN Classrooms ON Schedule.classroom_id = Classrooms.classroom_id
    WHERE
        Groups.group_name = p_group_name
        AND Schedule.week_number = p_week_number;

END;
$$ LANGUAGE plpgsql;

CREATE OR REPLACE FUNCTION update_student_avg_grade()
RETURNS TRIGGER AS $$
BEGIN
    UPDATE Students
    SET avg_grade = (
        SELECT AVG(grade)
        FROM Exams
        WHERE student_id = NEW.student_id
    )
    WHERE student_id = NEW.student_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER update_student_avg_grade_trigger
AFTER INSERT ON Exams
FOR EACH ROW
EXECUTE FUNCTION update_student_avg_grade();

-- Создание функции для логирования изменений в таблицу Logs
CREATE OR REPLACE FUNCTION log_students_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
        VALUES (
            'students',
            'INSERT',
```

```
                format(
                    'INSERT: student_id=%s, group_id=%s, user_id=%s, first_name=%s, last_name=%s, email=%s, phone_number=%s, avg_grade=%s',
                    NEW.student_id, NEW.group_id, NEW.user_id, NEW.first_name, NEW.last_name, NEW.email, NEW.phone_number, NEW.avg_grade
                )
            );

    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
        VALUES (
            'students',
            'UPDATE',
            format(
                'UPDATE: student_id=%s, group_id=%s, user_id=%s, first_name=%s, last_name=%s, email=%s, phone_number=%s, avg_grade=%s',
                NEW.student_id, NEW.group_id, NEW.user_id, NEW.first_name, NEW.last_name, NEW.email, NEW.phone_number, NEW.avg_grade
            )
        );

    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
            VALUES (
            'students',
            'DELETE',
            format(
                'DELETE: student_id=%s, group_id=%s, user_id=%s, first_name=%s, last_name=%s, email=%s, phone_number=%s, avg_grade=%s',
                NEW.student_id, NEW.group_id, NEW.user_id, NEW.first_name, NEW.last_name, NEW.email, NEW.phone_number, NEW.avg_grade
            )
        );
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Создание триггера для логирования изменений в таблицу Logs
CREATE TRIGGER students_changes_trigger
AFTER INSERT OR UPDATE OR DELETE
ON students
FOR EACH ROW
EXECUTE FUNCTION log_students_changes();


-- Создание функции для логирования изменений в таблицу Logs для курсов
CREATE OR REPLACE FUNCTION log_courses_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
        VALUES (
            'courses',
            'INSERT',
            format(
                'INSERT: course_id=%s, course_material_id=%s, course_name=%s, course_description=%s, number_of_hours=%s',
                NEW.course_id, NEW.course_material_id, NEW.course_name, NEW.course_description, NEW.number_of_hours
            )
        );

    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
        VALUES (
            'courses',
            'UPDATE',
            format(
                'UPDATE: course_id=%s, course_material_id=%s, course_name=%s, course_description=%s, number_of_hours=%s',
```

```
                NEW.course_id, NEW.course_material_id,
NEW.course_name, NEW.course_description, NEW.number_of_hours
            )
        );

    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
        VALUES (
            'courses',
            'DELETE',
            format(
                'DELETE: course_id=%s, course_material_id=%s,
course_name=%s, course_description=%s, number_of_hours=%s',
                NEW.course_id, NEW.course_material_id,
NEW.course_name, NEW.course_description, NEW.number_of_hours
            )
        );
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Создание триггера для логирования изменений в таблицу Logs для курсов
CREATE TRIGGER courses_changes_trigger
AFTER INSERT OR UPDATE OR DELETE
ON courses
FOR EACH ROW
EXECUTE FUNCTION log_courses_changes();


-- Создание функции для логирования изменений в таблицу Logs для
экзаменов
CREATE OR REPLACE FUNCTION log_exams_changes()
RETURNS TRIGGER AS $$
BEGIN
    IF TG_OP = 'INSERT' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
```

```
        VALUES (
            'exams',
            'INSERT',
            format(
                'INSERT: exam_id=%s, student_id=%s, course_id=%s, semester=%s, exam_date=%s, grade=%s',
                NEW.exam_id, NEW.student_id, NEW.course_id, NEW.semester, NEW.exam_date, NEW.grade
            )
        );

    ELSIF TG_OP = 'UPDATE' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
        VALUES (
            'exams',
            'UPDATE',
            format(
                'UPDATE: exam_id=%s, student_id=%s, course_id=%s, semester=%s, exam_date=%s, grade=%s',
                NEW.exam_id, NEW.student_id, NEW.course_id, NEW.semester, NEW.exam_date, NEW.grade
            )
        );

    ELSIF TG_OP = 'DELETE' THEN
        INSERT INTO Logs (table_name, action_type, action_description)
        VALUES (
            'exams',
            'DELETE',
            format(
                'DELETE: exam_id=%s, student_id=%s, course_id=%s, semester=%s, exam_date=%s, grade=%s',
                NEW.exam_id, NEW.student_id, NEW.course_id, NEW.semester, NEW.exam_date, NEW.grade
            )
        );
    END IF;
```

```
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

-- Создание триггера для логирования изменений в таблицу Logs для экзаменов
CREATE TRIGGER exams_changes_trigger
AFTER INSERT OR UPDATE OR DELETE
ON exams
FOR EACH ROW
EXECUTE FUNCTION log_exams_changes();
```

ПРИЛОЖЕНИЕ Б
(обязательное)
Конечная схема базы данных

ПРИЛОЖЕНИЕ В
(обязательное)
Ведомость документов