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

ФАКУЛЬТЕТ ИУ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА ИУ7 «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Разработка базы данных для хранения и обработки данных фитнес-клуба.

Студент	**ИУ7-64Б**		**Цховребова Я.Р.**
	(группа)	(подпись, дата)	(И.О. Фамилия)
Руководитель курсового проекта			**Гаврилова Ю.М.**
		(подпись, дата)	(И.О. Фамилия)
Консультант			
		(подпись, дата)	(И.О. Фамилия)

2025 г.

РЕФЕРАТ

Расчетно-пояснительная записка 56 с., 15 рис., 18 табл., 36 лист., 17 источн., 2 прил.

Ключевые слова: базы данных, фитнес-клуб, реляционная модель данных, Swift, PostgreSQL, Redis.

Целью данной курсовой работы является разработка базы данных для хранения и обработки данных фитнес-клуба.

Для достижения цели были выполнены следующие задачи:

— проведён анализ предметной области и формализована задача;

— описана структура базы данных и её пользователей;

— рассмотрены модели данных и выбрана наиболее подходящая;

— спроектирована база данных с реализацией триггеров для автоматического обновления данных;

— реализован интерфейс доступа к базе данных;

— проведено исследование производительности базы данных при различных нагрузках и с применением кеширования.

В результате выполненной работы спроектирована и реализована база данных для фитнес-клуба, обеспечивающая хранение и обработку информации, а также интерфейс для взаимодействия с системой.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последние годы наблюдается значительный рост интереса к физической активности и занятиям спортом, что, в свою очередь, привело к увеличению числа услуг, ориентированных на благополучие, здоровье и активный образ жизни, стремящихся удовлетворить растущий спрос и потребности общества [1].

Современные фитнес-клубы представляют собой не только места для занятий спортом, но и полноценные бизнес-структуры, в которых осуществляется широкий спектр процессов: регистрация и обслуживание клиентов, планирование и проведение тренировок, управление персоналом и тренерами, ведение финансовой отчетности и многое другое. В связи с этим возникает необходимость в создании надежной и гибкой информационной системы, основой которой является качественно спроектированная база данных.

Цель работы – разработать базу данных для хранения и обработки данных фитнес-клуба.

Для достижения цели курсовой работы необходимо выполнить следующие **задачи**:

1) провести анализ предметной области и формализовать задачу;
2) описать структуру базы данных и ее пользователей;
3) провести анализ моделей данных и выбрать наиболее подходящую;
4) спроектировать требуемую базу данных;
5) спроектировать триггеры для автоматического обновления данных;
6) выбрать средства реализации базы данных;
7) реализовать спроектированную базу данных и обеспечить её функциональность согласно проектным решениям;
8) реализовать интерфейс доступа к базе данных;
9) провести исследование производительности базы данных при увеличения объема данных, количества одновременных запросов, а также с использованием кеширование и без него.

1 Аналитическая часть

1.1 Анализ предметной области

Фитнес-клуб – это учреждение, оснащённое оборудованием для физических упражнений, предоставляющие услуги в области физической активности и здоровья [1, 2].

Цель фитнес-клуба – обеспечение комфортных условий для физической активности клиентов, улучшении их здоровья и поддержании высокого уровня физической формы [1, 2, 3].

Основные функции фитнес-клуба:

— обеспечение тренировок и активного отдыха для клиентов;

— предоставление абонементов с различными условиями для доступа к услугам клуба;

— управление расписанием тренировок;

— учет посещений, тренировки и прогресса клиентов [3].

Клиенты фитнес-клуба – физические лица, которые заинтересованы в поддержании физической активности и улучшении здоровья [2, 3].

Абонементы являются основным способом доступа клиентов к услугам фитнес-клуба, и их типы определяются самим клубом в зависимости от потребностей и предпочтений клиентов. Например, могут быть предложены ежемесячные абонементы (предоставляющие доступ на один месяц) либо годовые абонементы (с выгодными условиями на длительный период) [3].

Основными **сотрудниками** фитнес-клуба являются:

— **администраторы**, которые управляют клиентской базой, занимаются продажей абонементов, отслеживанием посещаемости и предоставляют информацию о клубе.

— **тренеры** – специалисты, которые проводят тренировки для клиентов [3].

1.2 Обзор существующих решений

1С:Фитнес клуб

1С:Фитнес клуб – это программное решение для автоматизации фитнес-клубов и спортивных учреждений, созданное компанией «Лаборатория программного обеспечения».

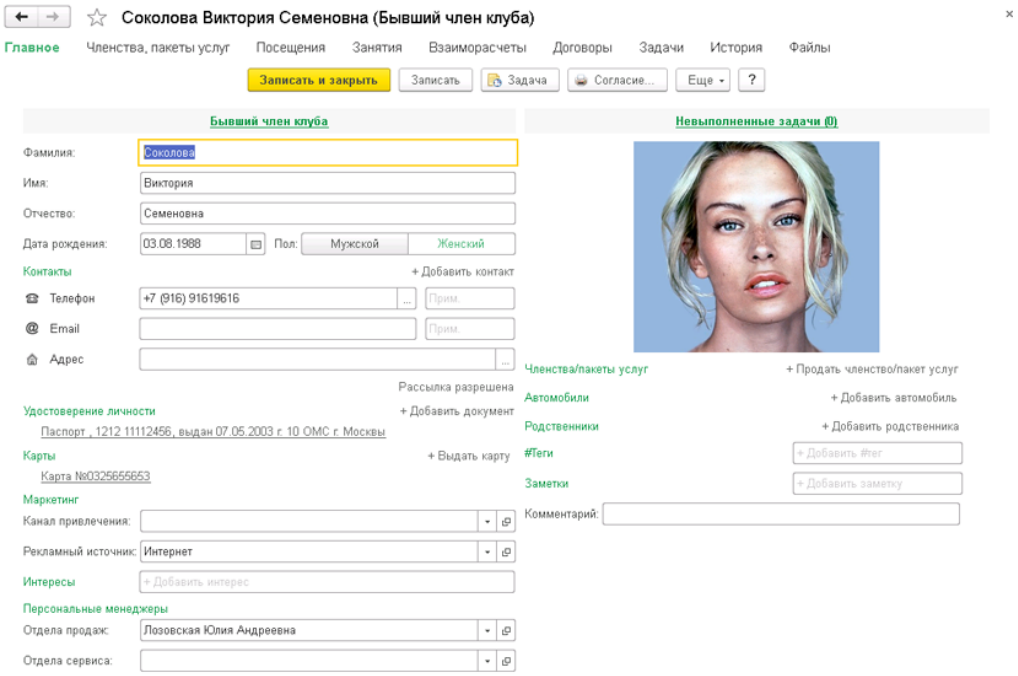

Рисунок 1 – Интерфейс администратора для управления данными клиента фитнес-клуба приложения 1С:Фитнес клуб

Приложение включает в себя функционал для работы с клиентской базой, маркетинга, расчета заработной платы, учета посещений, CRM-систему, интеграции с внешними сервисами и мобильные приложения для персонала.

Приложение не предоставляет пробный период, но предлагает несколько тарифных планов, подходящих для разных масштабов бизнеса. Тарифы варьируются от 1 990 рублей в месяц за облачную подписку с базовым набором функций до 90 000 рублей за покупку программы с расширенными возможностями.

Для работы приложения необходима операционная система Windows. Для онлайн версии требуется постоянный интернет на компьютере.

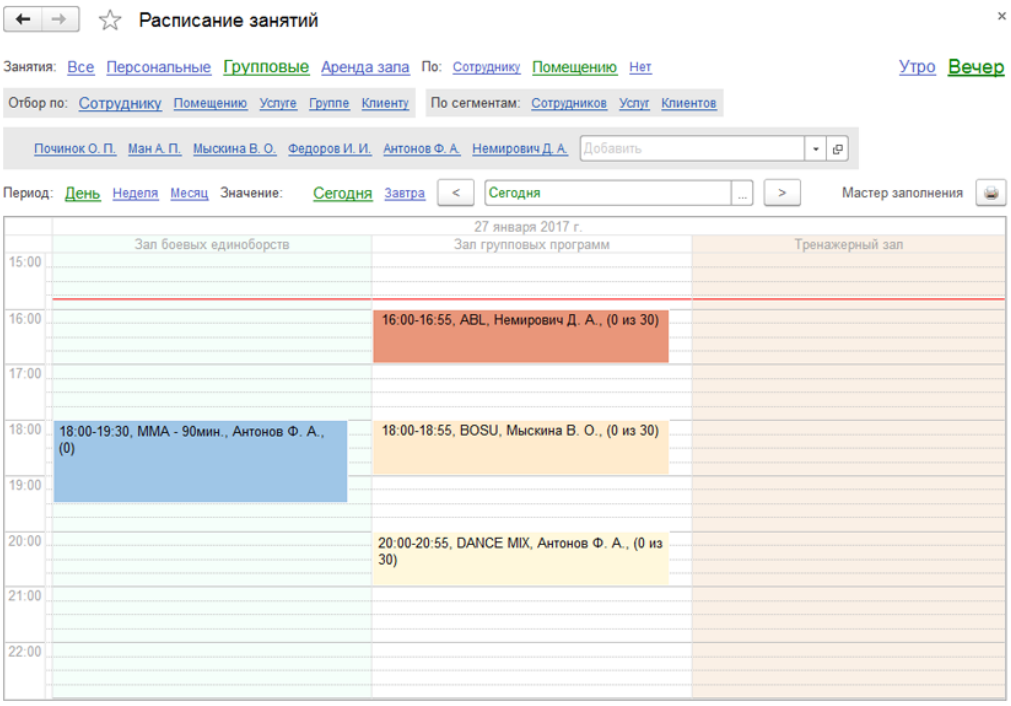

Рисунок 2 – Интерфейс администратора для управления расписание тренировок фитнес-клуба приложения 1С:Фитнес клуб

fitness365

fitness365 – это веб-система, специально разработанная для комплексного управления фитнес-клубом. Она предоставляет широкий набор инструментов, которые охватывают все аспекты работы клуба: от работы с клиентами до ведения статистики и аналитики.

fitness365 имеет интегрированную CRM-систему, которая позволяет клубу управлять базой клиентов – каждому пользователю предоставляется персонализированный доступ.

Тарифы fitness365 предлагают различные опции для фитнес-клубов. Например, тариф «Онлайн» стоит 2 000 рублей в месяц, включает доступ через интернет и возможность работы с неограниченным числом клиентов, но поддерживает 5 рабочих мест. Для клубов, где могут быть проблемы с интернетом, доступна настольная версия за разовый платеж 44 000 рублей.

Для работы приложения необходима операционная система Windows, для онлайн версии требуется постоянный интернет на компьютере.

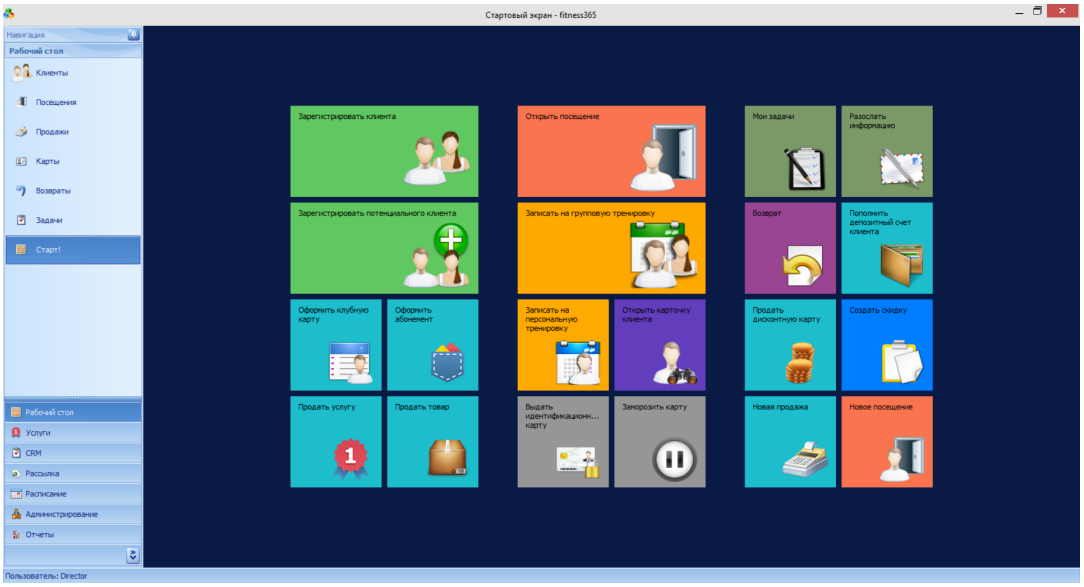

Рисунок 3 – Интерфейс главной страницы администратора фитнес-клуба приложения fitness365

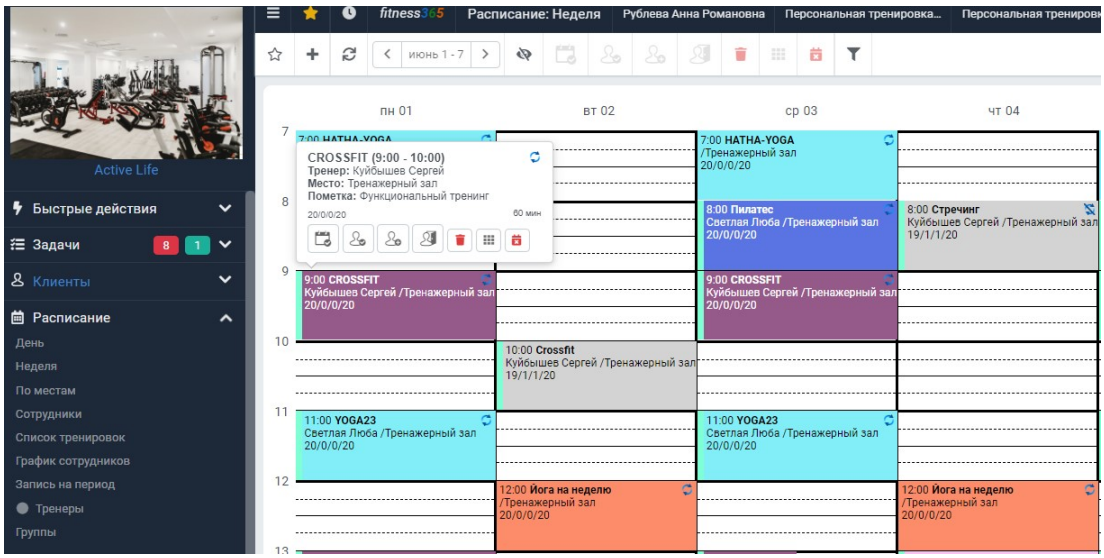

Рисунок 4 – Интерфейс администратора для управления расписанием фитнес-клуба приложения fitness365

Сравнение решений

Рассматриваемые решения, такие как 1С:Фитнес Клуб и fitness365, могут быть избыточными для использования в условиях небольших или специализированных клубов. Кроме того, все рассматриваемые решения являются платными, что может стать препятствием для небольших фитнес-клубов.

Разрабатываемая база данных будет сфокусирована для работы клю-

Функциональность для администратора	**1С:Фитнес Клуб**	**fitness365**
Управление расписанием, залами и персоналом	+	+
Работа с абонементами и услугами	+	+
Ведение клиентской базы и CRM	+	+
Функциональность для клиента	**1С:Фитнес Клуб**	**fitness365**
Наличие личного кабинета	+	+
Покупка абонементов онлайн	+	+
Самостоятельная запись на тренировки	+	+
Функциональность для тренера	**1С:Фитнес Клуб**	**fitness365**
Доступ к информации о клиентах	–	+
Просмотр и управление своим расписанием	–	–
Мобильный доступ к системе	–	–

Таблица 1 – Сравнение функциональности по 1С:Фитнес Клуб и fitness365

чевых функций, таких как управление расписанием для тренеров и клиентов, самостоятельный контроль личного кабинета. Кроме того, программное обеспечение будет бесплатным, что делает его более доступным для широкого круга пользователей.

1.3 Формализация данных

На рисунке 5 изображена диграмма «сущность–связь» в нотации Чена.

В базе данных для фитнес-клуба можно выделить следующие ключевые сущности.

1. **Пользователь** – основная сущность, представляющая всех участников системы: клиентов, тренеров и администраторов.

2. **Тренер** – сущность, представляющая пользователя, проводящего тренировки.

3. **Специализация** – сущность, которая определяет специализацию и используется для описания тренировок и компетенций тренеров.

4. **Тип абонемента** – сущность, описывающая варианты абонементов, устанавливаемые фитнес-клубом.

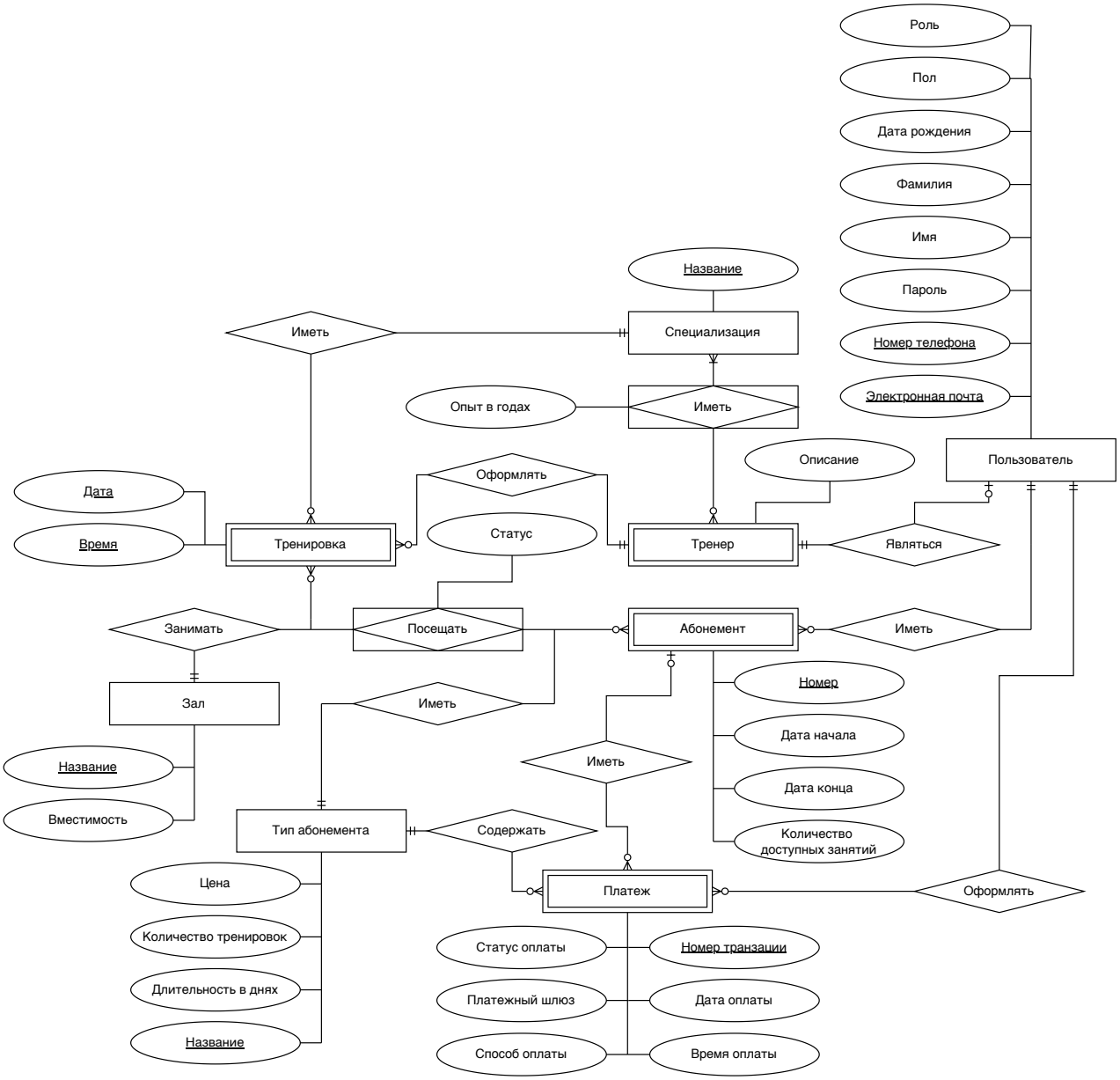

Рисунок 5 – Диаграмма «сущность-связь» фитнес-клуба в нотации Чена

5. **Абонемент** – сущность, отражающая приобретённый пользователем абонемент.

6. **Платеж** – сущность, отражающая факт оплаты заказа.

7. **Зал** – сущность, которая содержит информацию о помещениях для проведения тренировок.

8. **Тренировка** – сущность, представляющая собой запланированное мероприятие – тренировку.

9. **Посещение** – сущность, которая фиксирует факт участия клиента в конкретной тренировке.

1.4 Формализация пользователей и их прав доступа

Взаимодействовать с базой данных будут четыре вида пользователей.

На рисунке 6 приведена диаграмма вариантов использования базы данных.

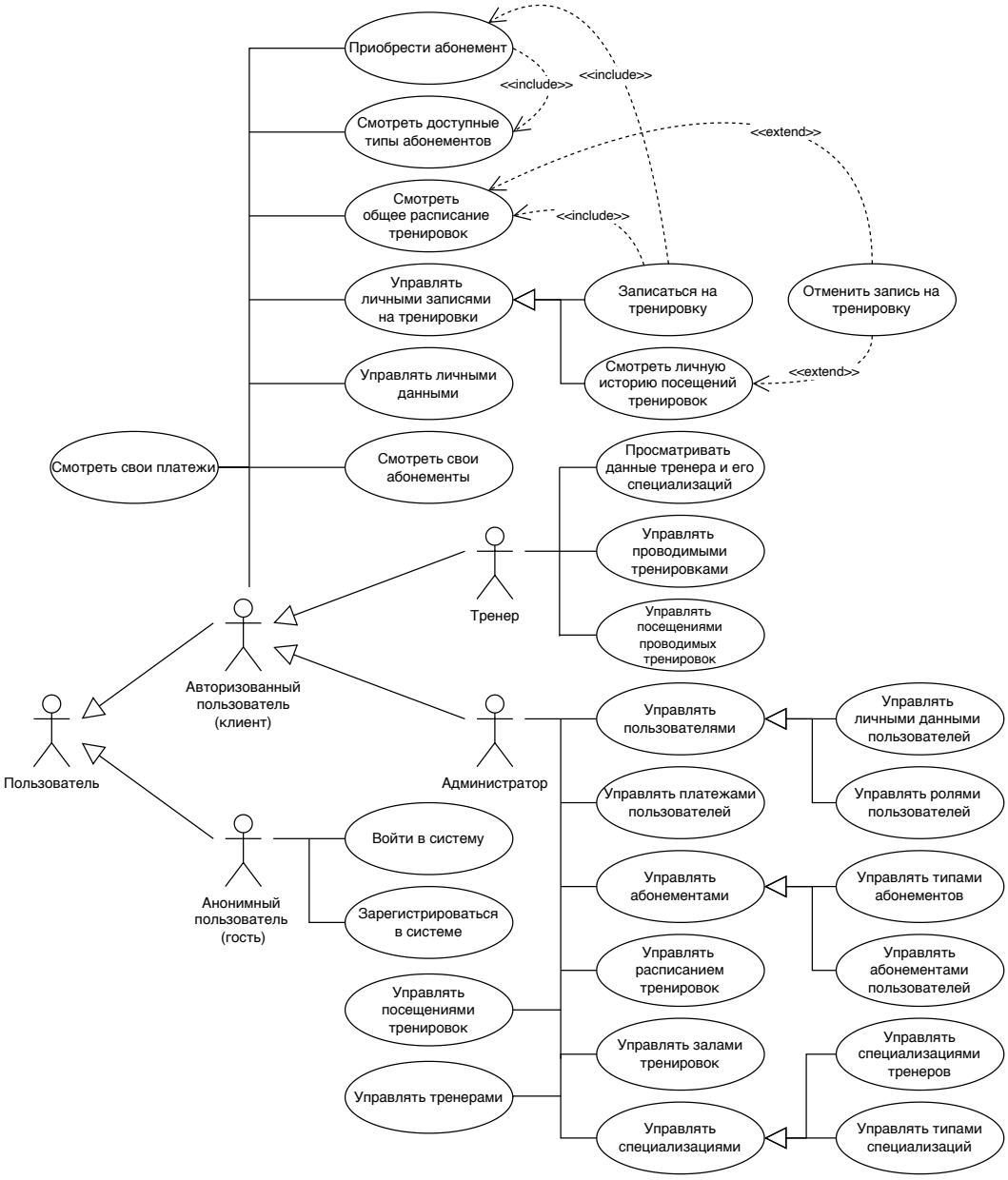

Рисунок 6 – Диаграмма вариантов использования базы данных

Гость – анонимный пользователь, который может выполнять только определенные действия, такие как регистрация и вход в систему.

Авторизованные пользователи:

— **клиент** – пользователь, который имеет минимальные права, ограниченные только возможностью взаимодействовать с данными, относящимися к его учетной записи и обслуживанию (например, заказами, абонементами и посещениями);

— **тренер** – пользователь, имеющий права клиента, а также имеющий доступ к данным, связанным с его тренерской деятельностью;

— **администратор** – пользователь, который имеет полный доступ к любым данным.

1.5 Модели данных

Модель данных представляет собой формализованное описание структуры информационных единиц и операций с ними в информационной системе, которые определяет логическую организацию базы данных и способы хранения, организации и обработки данных [4, с. 4].

Существует множество моделей данных, которые делятся на дореляционные(иерархические, сетевые, основнные на инвертированных списках), реляционные, постреляционные модели (например, ключ-значение, столбцовые, документные и графовые). Каждая из этих моделей имеет свои особенности и применяется в различных областях [4].

Основываясь на рейтинге из [5], в дальнейшем будут рассмотрены наиболее популярные в настоящее время модели данных.

1.5.1 Реляционная модель данных

Реляционная модель была предложена Э. Коддом в 1970 году в статье [6] и основана на теории отношений, опирается на математическое понятие n-арного отношения, что представляет собой подмножество декартового произведения.

Основными понятия реляционных баз данных являются тип данных, домен, атрибут, кортеж, отношение, первичный ключ.

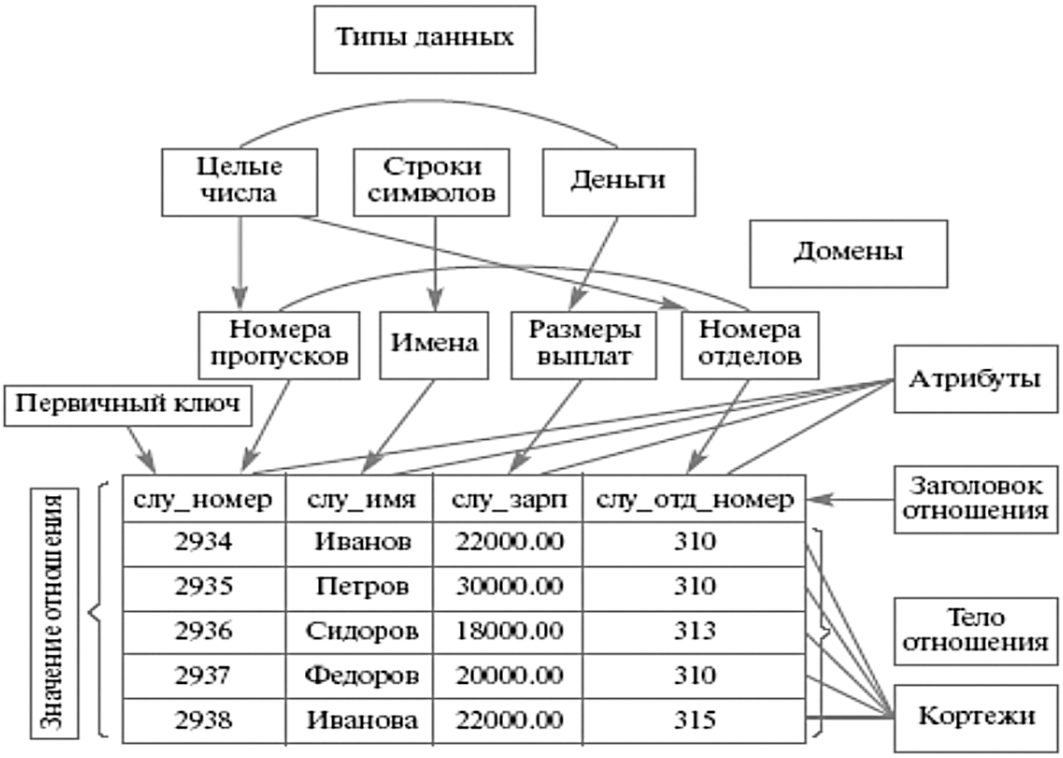

Рисунок 7 – Основные понятия реляционной модели данных [4, с. 31]

Атрибут – некоторая характеристика объекта (сущности), имеющая уникальное имя внутри отношения, через которое к ниму производится обращение.

Кортеж – это совокупность значений, где каждое значение (или элемент) соответствует определенному атрибуту, и эти значения принадлежат соответствующему домену атрибута.

Множество всех кортежей образует **отношение**, которое является подмножеством декартового произведения доменов, причем количество кортежей в отношении называется *мощностью отношения.*

Схема отношения (заголовок отношения) – набор упорядоченных пар <A, T>, где A – имя атрибута, а T – домен. Количество атрибутов в схеме называется *степенью отношения.*

Таким образом, **реляционная база данных** – это множество отношений, представленных в виде таблиц, где заголовок – схема отношения, а строки – кортежи.

Реляционная модель данных, по К. Дейту, состоит из трех частей.

1. **Структурная часть** – описывает организацию данных, включая таб-

лицы (отношения), атрибуты и их типы, а также связи между таблицами.

2. **Манипуляционная часть** – включает операции над данными, такие как выборка, добавление, обновление и удаление данных, что осуществляется через язык запросов (например, SQL).

3. **Целостная часть** – обеспечивает соблюдение целостности данных, включая ограничения (например, уникальность значений, ссылки между таблицами через внешние ключи), чтобы поддерживать корректность и непротиворечивость данных в базе [4, С. 30-35].

1.5.2 Модель данных «ключ-значение»

Модель данных ключ-значение представляет собой структуру данных, которая использует ассоциативный массив, где каждый элемент состоит из *уникального ключа* и *связанного с ним значения*. Значение может быть любым типом данных, но оно не имеет структуры.

Эта модель поддерживает **две основные операции**:

— **получение значения по ключу** – если ключ существует, возвращается связанное с ним значение, иначе – NULL (специальное значение, которое используется в базах данных для обозначения отсутствия данных или неизвестного значения).

— **запись значения по ключу** – позволяет добавить или обновить значение для определенного ключа (также возможно установить время жизни ключа, после чего он будет автоматически удален) [4, С. 89-91].

1.5.3 Документная модель данных

Модель документная расширяет представление модели «ключ-значение», позволяя хранить более сложные структуры данных – документы.

Документная модель данных – это подход к организации и хранению данных, при котором данные представлены в виде документов.

Основные характеристики документной модели:

1) документ как основная единица хранения;

2) **документ** – это структурированный набор данных, который может содержать различные пары *ключ-значение* и может включать

   — простые типы данных: строки, числа, булевы значения;

   — упорядоченные списки значений;

   — вложенные документы: другие объекты, которые могут быть представлены в формате пары ключ-значение;

   — сложные типы данных: например, даты, бинарные данные и другие;

3) отсутствие операций соединения – связанные данные обычно хранятся в одном документе [4, С. 92-96].

Выбор модели данных

В таблице 2 представлено сравнение трёх моделей данных – реляционной модели (РМ), модели ключ–значение (КЗМ) и документной модели (ДМ). Для удобства использованы условные обозначения: «+» – высокая степень поддержки критерия, «-» — низкая или отсутствующая поддержка, «+/-» — частичная или ограниченная поддержка.

Критерий	**РМ**	**КЗМ**	**ДМ**
Поддержка связей между сущностями	+	-	+/-
Структурированность данных	+	-	+/-
Масштабируемость	+/-	+	+

Таблица 2 – Сравнение моделей данных для базы данных фитнес-клуба

Для базы данных фитнес-клуба выбрана реляционная модель ввиду её строгой структурированности и поддержки связей между сущностями.

Для кэширования выбрана модель ключ–значение, обеспечивающая гибкость и масштабируемость при работе с простыми данными, не требующими сложной структуры и поддержки связей, что делает документную модель избыточной для данной задачи.

1.6 Формализация задачи

Предметная область охватывает деятельность фитнес-клуба, включая управление данными о клиентах (их личной информации, тренировках, истории посещений), тренерах (их расписаниях, специализациях) и организации тренировок, расписаний, а также систему учета абонементов.

Цель создания базы данных – автоматизация учёта и повышение эффективности работы фитнес-клуба и качества обслуживания клиентов.

Для взаимодействия с базой данных фитнес-клуба необходимо разработать интерфейс, который обеспечит функциональный доступ к ключевым операциям системы в зависимости от роли пользователя.

Интерфейс должен предусматривать следующие возможности:

— регистрация и авторизация пользователей;

— приобретение абонементов;

— запись на тренировки и управление расписанием.

В рамках данной курсовой работы не рассматривается вопрос обеспечения конфиденциальности персональных данных пользователей и интеграции платежной системы:

— все данные, включая контактные и учетные данные пользователей, будут храниться в базе данных без использования специализированных механизмов защиты конфиденциальности;

— не реализуется функциональность для работы с платежными системами и интеграция с внешними платёжными сервисами для обработки финансовых транзакций или хранения данных о платежах.

Вывод

В данном разделе проведен анализ предметной области фитнес-клубов, описана структура базы данных, рассмотрены пользователи базы данных и их права доступа. Также проведен анализ различных моделей данных, в результате чего для хранения данных выбрана реляционная модель данных, а для кэширования – модель «ключ-значение». Рассмотрены существующие решения и их особенности.

2 Конструкторская часть

В данном разделе спроектирована структура базы данных для фитнес-клуба, включая описание сущностей, ограничений целостности данных, а также модели ролей пользователей. Также спроектированы функции, процедуры и триггеры, обеспечивающие автоматизацию обновления данных.

2.1 Разработка базы данных

На рисунке 8 представлена схема проектируемой базы данных.

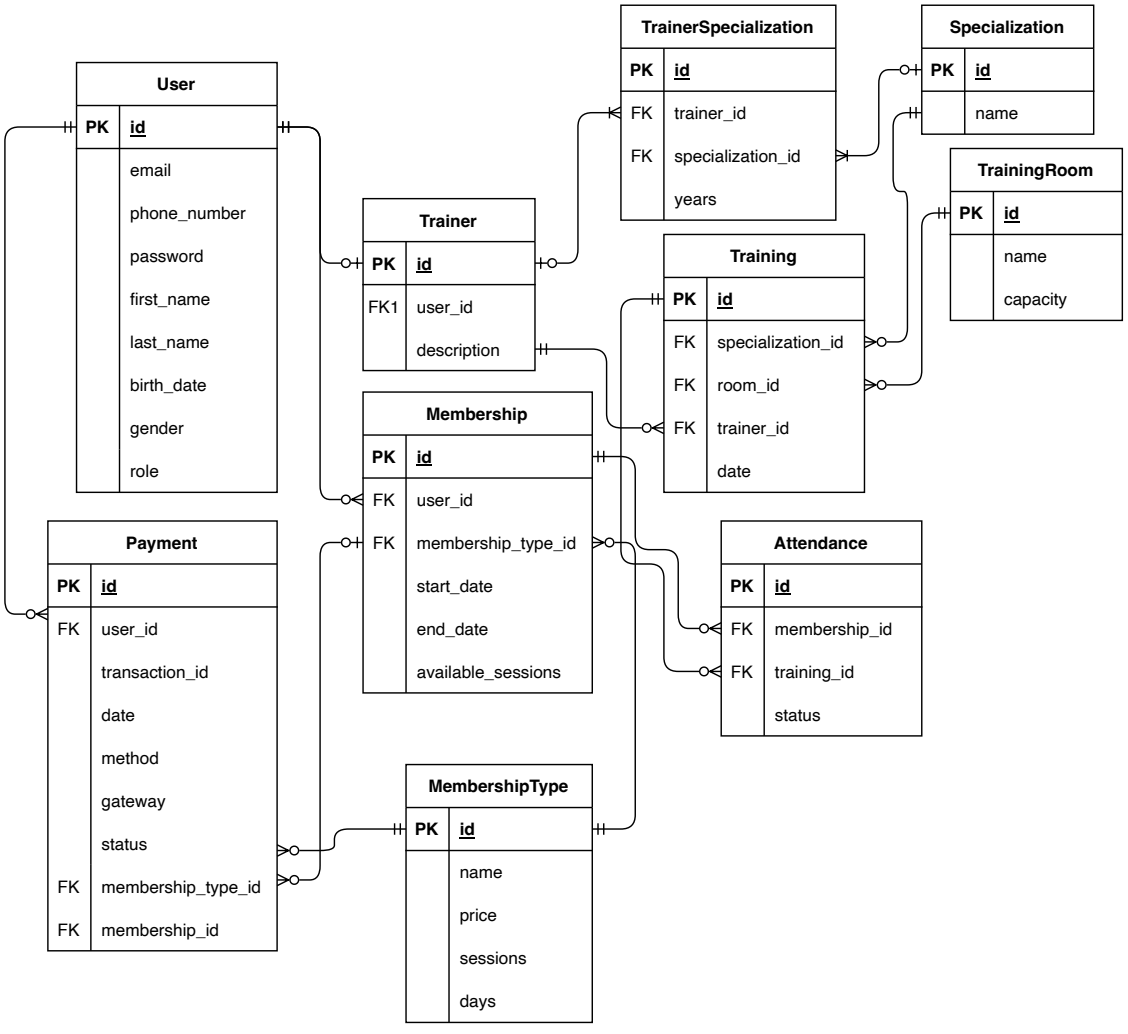

Рисунок 8 – Диграмма «сущность-связь» базы данных фитнес-клуба

2.2 Разработка сущностей базы данных

Далее будут описаны сущности проектируемой базы данных.

User – сущность, представляющая зарегистрированного пользователя системы.

Атрибуты:

— *id* – уникальный идентификатор пользователя;

— *first_name* – имя;

— *last_name* – фамилия;

— *email* – адрес электронной почты;

— *password* – пароль;

— *phone_number* – номер телефона;

— *birth_date* – дата рождения;

— *role* – роль;

— *gender* – пол.

Trainer – сущность, представляющая дополнительная информацию о пользователях-тренерах.

Атрибуты:

— *id* – уникальный идентификатор тренера;

— *user_id* – внешний ключ, ссылающийся на пользователя;

— *description* – краткое описание, биография.

Specialization – сущность, представляющая типы специализаций тренеров и тренировок.

Атрибуты:

— *id* – уникальный идентификатор специализации;

— *name* – название специализации.

TrainerSpecialization – сущность, представляющая связь между тренерами и их специализациями, специализации тренеров.

Атрибуты:

— *id* – уникальный идентификатор специализации тренера;

— *trainer_id* – внешний ключ, ссылающийся на тренера;

— *specialization_id* – внешний ключ, ссылающийся на специализацию;

— *years* – опыт тренера в годах для специализации.

TrainingRoom – сущность, представляющая залы, в которых проводятся тренировки.

Атрибуты:

— *id* – уникальный идентификатор зала;

— *name* – название зала;

— *capacity* – вместимость зала.

MembershipType – сущность, представляющая типы абонементов, доступных для покупки.

Атрибуты:

— *id* – уникальный идентификатор типа абонемента;

— *name* – название типа абонемента;

— *days* – длительность действия в днях;

— *price* – стоимость;

— *sessions* – количество доступных тренировок.

Membership – сущность, представляющая абонементы, приобретённые пользователями.

Атрибуты:

— *id* – уникальный идентификатор абонемента;

— *user_id* – внешний ключ, ссылающийся на пользователя-клиента;

— *start_date* – дата начала действия;

— *end_date* – дата окончания действия;

— *available_sessions* – количество доступных (оставшихся) тренировок.

Payment – сущность, представляющая информацию об оплате абонемента.

Атрибуты:

— *id* – уникальный идентификатор платежа;

— *membership_id* – внешний ключ, ссылающийся на абонемент;

— *membership_type_id* – внешний ключ, ссылающийся на тип абонемента;

— *transaction_id* – идентификатор транзакции, получаемый от платежной системы;

— *date* – дата платежа;

— *method* – способ оплаты;

— *gateway* – платежный шлюз;

— *status* – статус оплаты.

Training – сущность, представляющая тренировки.

Атрибуты:

— *id* – уникальный идентификатор тренировки;

— *trainer_id* – внешний ключ, ссылающийся на тренера;

— *room_id* – внешний ключ, ссылающийся на зал;

— *specialization_id* – внешний ключ, ссылающийся на специализацию тренировки.

Attendance – сущность, представляющая записи на тренировку пользователей.

Атрибуты:

— **id** – уникальный идентификатор записи на тренировку;

— **training_id** – внешний ключ, ссылающийся на тренировку;

— **user_id** – внешний ключ, ссылающийся пользователя-клиента;

— **status** – статус участия.

2.3 Разработка ограничений целостности данных

Далее в таблицах 3–12, представлены ограничения целостности данных, разработанные для сущностей проектируемой базы данных.

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
user_id	UUID	Внешний ключ (ссылается на User), уникальный, каскадное удаление
description	Строка	Не NULL, длина до 511 символов

Таблица 3 – Ограничения атрибутов сущности Trainer

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
name	Строка	Не NULL, длина до 127 символов, уникальное

Таблица 4 – Ограничения атрибутов сущности Specialization

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
name	Строка	Не NULL, длина до 63 символов, уникальное
capacity	Целое число	Не NULL, больше 0

Таблица 5 – Ограничения атрибутов сущности TrainingRoom

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
name	Строка	Не NULL, уникальное, длина до 127 символов
price	Число с фиксированной точностью	Не NULL, больше или равно 0.0, по умолчанию 0.0
sessions	Целое число	Не NULL, больше 0, по умолчанию 1
days	Целое число	Не NULL, больше 0, по умолчанию 1

Таблица 6 – Ограничения атрибутов сущности MembershipType

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
membership_id	UUID	Внешний ключ (ссылается на Membership), каскадное удаление
training_id	UUID	Внешний ключ (ссылается на TrainingSession), при удалении родительской записи в связанной таблице поле устанавливается в NULL
status	Строка	Не NULL, длина до 63 символов, значения: ('посетил', ожидает', 'отсутствовал'), по умолчанию 'ожидает'

Таблица 7 – Ограничения атрибутов сущности Attendance

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
email	Строка	Не NULL, уникален, проверка по регулярному выражению, которое соответствует формату email (например, user@example.com), длина до 127 символов
phone_number	Строка	Не NULL, уникален, проверка по регулярному выражению, которое соответствует телефонным номерам в международном формате (например, +1234567890), длина до 31 символов
password	Строка	Не NULL, длина до 255 символов
first_name	Строка	Не NULL, проверка по регулярному выражению, которое позволяет только буквы (с возможным дефисом внутри имени), длина до 127 символов
last_name	Строка	Не NULL, проверка по регулярному выражению, которое позволяет только буквы (с возможным дефисом внутри фамилии), длина до 127 символов
birth_date	Дата	Не NULL, проверка на возраст: от 14 до 120 лет
gender	Строка	Не NULL, значения: ('мужской', 'женский'), длина до 31 символов
role	Строка	Не NULL, значения – ('клиент', 'тренер', 'администратор'), длина до 31 символов

Таблица 8 – Ограничения атрибутов сущности User

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
trainer_id	UUID	Внешний ключ (ссылается на Trainer), при удалении родительской записи в связанной таблице поле устанавливается в NULL
room_id	UUID	Внешний ключ (ссылается на TrainingRoom), при удалении родительской записи в связанной таблице поле устанавливается в NULL
specialization_id	UUID	Внешний ключ (ссылается на Specialization), при удалении родительской записи в связанной таблице поле устанавливается в NULL
date	Метка времени	Не NULL

Таблица 9 – Ограничения атрибутов сущности Training

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
trainer_id	UUID	Внешний ключ (ссылается на Trainer), каскадное удаление
specialization_id	UUID	Внешний ключ (ссылается на Specialization), уникальная комбинация (trainer_id, specialization_id), каскадное удаление
years	Целое число	Не NULL, больше или равно 0

Таблица 10 – Ограничения атрибутов сущности TrainerSpecialization

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
user_id	UUID	Внешний ключ (ссылается на User), при удалении родительской записи в связанной таблице поле устанавливается в NULL
transaction_id	Строка	Не NULL, уникальное, длина до 255 символов
date	Метка времени	Не NULL, по умолчанию текущая Метка времени
method	Строка	Не NULL, значения – ('наличные', 'кредитная карта', 'банковский перевод'), длина до 63 символов, по умолчанию 'наличные'
gateway	Строка	Можеть быть NULL, длина до 63 символов, по умолчанию NULL
status	Строка	Не NULL, длина до 63 символов, значения: ('ожидает', 'оплачен', 'отменен'), по умолчанию 'ожидает'.
membership_type_id	UUID	Внешний ключ (ссылается на MembershipType), при удалении родительской записи в связанной таблице поле устанавливается в NULL
membership_id	UUID	Внешний ключ (ссылается на Membership), при удалении родительской записи в связанной таблице поле устанавливается в NULL

Таблица 11 – Ограничения атрибутов сущности Payment

Артибут	**Тип данных**	**Ограничение**
id	UUID	Уникальный идентификатор
user_id	UUID	Внешний ключ (ссылается на User), каскадное удаление
membership_type_id	UUID	Внешний ключ (ссылается на MembershipType), при удалении родительской записи в связанной таблице поле устанавливается в NULL
start_date	Дата	Может быть NULL, по умолчанию NULL
end_date	Дата	Может быть NULL, по умолчанию NULL, не раньше start_date
available_sessions	Целое число	Не NULL, больше или равно 0, по умолчанию 0

Таблица 12 – Ограничения атрибутов сущности Membership

2.4 Разработка ролевой модели

Для обеспечения безопасности и разграничения доступа к данным разработанной базе данных фитнес-клуба необходимо разработать ролевую модель управления доступом.

Роли базы данных

1. **Гость**
   - имеет минимальные права;
   - имеет доступ только к функциям регистрации и аутентификации;
   - не имеет прямого доступа к таблицам базы данных.
2. **Клиент**
   - является зарегистрированным и аутентифицированным пользователем системы;

— может приобретать абонементы и имеет возможность записываться на тренировки;

— имеет доступ только к собственным записям в таблицах *User*, *Membershi* *Payment*, *Attendance*;

— может просматривать данные, содержащуюся в таблицах *Specialization*, *TrainerSpecialization*, *MembershipType*, *Trainer*, *TrainingRoom*, *Training*.

3. **Тренер**

— наследует все права клиента, но обладает расширенными возможностями;

— имеет доступ к записям в таблицах *Training* и *Attendance*, где он указан в качестве тренера;

— может просматривать информацию о пользователях и их посещениях тренировок;

— имеет право изменять данные только тех тренировок (включая посещения этих тренировок), которые он проводит.

4. **Администратор** обладает полными правами на управление базой данных.

2.5 Разработка функции, процедур и триггеров

Триггер, алгоритм которого представлен на рисунке 9, предназначен для автоматического обновления количества доступных тренировок у абонемента пользователя в таблице Membership в зависимости от изменения статуса посещения тренировки в таблице Attendance.

Триггер, алгоритм которого представлен на рисунке 10, предназначен для проверки вместимости зала перед тем, как записать пользователя на тренировку. Он предотвращает запись, если в зале нет свободных мест для выбранной тренировки.

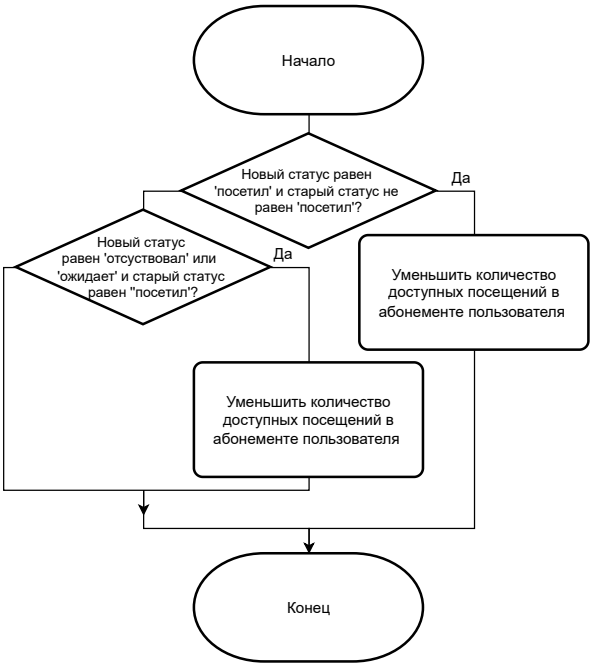

Рисунок 9 – Схема алгоритма работы триггера для обновления данных после изменения статуса посещения тренировки

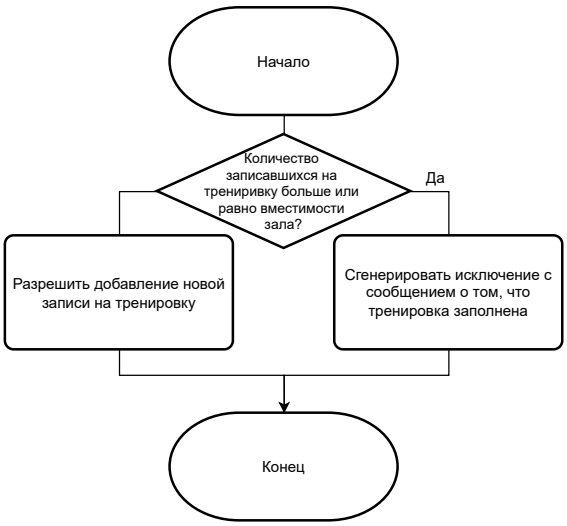

Рисунок 10 – Схема алгоритма работы триггера для проверки данных перед добавлением записи на тренировку

Вывод

В данном разделе были разработаны база данных, ее сущности и ограничения целостности данных, а также спроектированы триггеры, обеспечивающие автоматическое обновление данных.

3 Технологическая часть

3.1 Средства реализации

В таблице 13 представлено сравнение четырёх наиболее популярных реляционных систем управления базами данных (СУБД), таких как PostgreSQL, MySQL, Microsoft SQL Server и Oracle Database [5]. Для удобства использованы условные обозначения: «+» – высокий уровень соответствия критерию, «-» – низкий уровень соответствия, «+/-» – частичное соответствие критерию с наличием определённых ограничений.

Критерий	**PostgreSQL**	**MySQL**	**MS SQL Server**	**Oracle**
Поддержка надёжности и ACID-свойств [7]	+	+/-	+	+
Совокупная стоимость владения	+	+	+/-	-

Таблица 13 – Сравнительный анализ реляционных СУБД для информационной системы фитнес-клуба

В качестве СУБД выбрана **PostgreSQL** [8]. Данная система поддерживает расширенные механизмы обеспечения целостности данных и предоставляет гибкие средства безопасности, включая разграничение доступа на уровне ролей и строк, что существенно повышает уровень защиты информации и позволяет реализовать строгие политики доступа. Особым преимуществом является то, что PostgreSQL распространяется по лицензии с открытым исходным кодом и не требует лицензионных платежей, что значительно снижает совокупную стоимость владения системой и делает её экономически выгодным решением для различных проектов. Кроме того, имеется практический опыт работы с этой системой, что снижает риски при внедрении.

Для кэширования данных используется **Redis** [9] – является популярным решением [5].

Для разработки интерфейса доступа выбран язык программирования **Swift** [10], так как он предоставляет все необходимые инструменты для эффективной работы с PostgreSQL и создания удобного пользовательского интерфейса. Кроме того, имеется практический опыт работы с этим языком.

В качестве средств разработки выбраны **Xcode** [11] и **pgAdmin 4** [12]. Xcode обеспечивает полноценную среду для разработки интерфейсов на Swift, а pgAdmin 4 предоставляет все необходимые инструменты для управления базой данных PostgreSQL.

3.2 Реализация базы данных

Далее представлены реализации спроектированной базы данных: ее сущностей, ограничений целостности данных, ролевой модели и триггеров.

Реализация сущностей

Реализации создания таблиц представлены на листингах 1-10.

Листинг 1 – Реализация создания отношения Trainer

```
1     CREATE TABLE "Trainer" (
2     id                UUID,
3     user_id           UUID,
4     description       TEXT
5     );
```

Листинг 2 – Реализация создания отношения User

```
1  CREATE TABLE "User" (
2      id                UUID,
3      email             TEXT,
4      phone_number      TEXT,
5      "password"        TEXT,
6      first_name        TEXT,
7      last_name         TEXT,
8      gender            TEXT,
9      birth_date        DATE,
10     "role"            TEXT
11 );
```

Листинг 3 – Реализация создания отношения Specialization

```
1  CREATE TABLE "Specialization" (
2      id        UUID,
3      name      TEXT
4  );
```

Листинг 4 – Реализация создания отношения TrainerSpecialization

```
CREATE TABLE "TrainerSpecialization" (
    id                  UUID,
    trainer_id          UUID,
    specialization_id   UUID,
    years               INT
);
```

Листинг 5 – Реализация создания отношения MembershipType

```
CREATE TABLE "MembershipType" (
    id          UUID,
    name        TEXT,
    price       NUMERIC,
    sessions    INT,
    days        INT
);
```

Листинг 6 – Реализация создания отношения TrainingRoom

```
CREATE TABLE "TrainingRoom" (
    id          UUID,
    name        TEXT,
    capacity    INT
);
```

Листинг 7 – Реализация создания отношения Payment

```
CREATE TABLE "Payment" (
    id                  UUID,
    user_id             UUID,
    membership_id       UUID,
    membership_type_id  UUID,
    transaction_id      TEXT,
    "date"              TIMESTAMP,
    "method"            TEXT,
    gateway             TEXT,
    status              TEXT
);
```

Листинг 8 – Реализация создания отношенияMembership

```
CREATE TABLE "Membership" (
    id                  UUID,
    user_id             UUID,
    membership_type_id  UUID,
    start_date          DATE,
    end_date            DATE,
    available_sessions  INT
);
```

Листинг 9 – Реализация создания отношения Training

```
CREATE TABLE "Training" (
    id                  UUID,
    specialization_id   UUID,
    room_id             UUID,
    trainer_id          UUID,
    "date"              TIMESTAMP
);
```

Листинг 10 – Реализация создания отношения Attendance

```
    CREATE TABLE "Attendance" (
    id              UUID,
    membership_id   UUID,
    training_id     UUID,
    status          TEXT
);
```

Реализация ограничений целостности данных

Реализации ограничений целостности данных таблиц представлены на листингах 11-20.

Листинг 11 – Реализация ограничений целостности данных отношения Specialization

```
ALTER TABLE "Specialization"
    ADD CONSTRAINT "pk:specialization.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "chk:specialization.name:notnull" CHECK (
    name IS NOT NULL),
    ADD CONSTRAINT "chk:specialization.name:length" CHECK (
    length(name) < 128),
    ADD CONSTRAINT "uq:specialization.name" UNIQUE (name);
```

Листинг 12 – Реализация ограничений целостности данных отношения User

```
ALTER TABLE "User"
    ADD CONSTRAINT "pk:user.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "chk:user.email:notnull" CHECK (email IS NOT NULL),
    ADD CONSTRAINT "chk:user.email:length" CHECK (length(email) < 128),
    ADD CONSTRAINT "chk:user.email:regexp" CHECK (
        email ~ '^[A-Za-z0-9._%-]+@[A-Za-z0-9.-]+[.][A-Za-z]+$'),
    ADD CONSTRAINT "uq:user.email" UNIQUE (email),
    ADD CONSTRAINT "chk:user.phone_number:notnull" CHECK (
        phone_number IS NOT NULL),
    ADD CONSTRAINT "chk:user.phone_number:length" CHECK (
        length(phone_number) < 32),
    ADD CONSTRAINT "chk:user.phone_number:regexp" CHECK (
        phone_number ~ '^\+\d{10,15}$'),
    ADD CONSTRAINT "uq:user.phone_number" UNIQUE (phone_number),
    ADD CONSTRAINT "chk:user.password:notnull" CHECK (
        "password" IS NOT NULL),
    ADD CONSTRAINT "chk:user.first_name:notnull" CHECK (
        first_name IS NOT NULL),
    ADD CONSTRAINT "chk:user.first_name:length" CHECK (
        length(first_name) < 128),
    ADD CONSTRAINT "chk:user.first_name:regexp" CHECK (
        first_name ~ '^([a-zA-Za-яА-ЯёЁ]+(?:-[a-zA-Za-яА-ЯёЁ]+)?)$'),
    ADD CONSTRAINT "chk:user.last_name:notnull" CHECK (
        last_name IS NOT NULL),
    ADD CONSTRAINT "chk:user.last_name:length" CHECK (
        length(last_name) < 128),
    ADD CONSTRAINT "chk:user.last_name:regexp" CHECK (
        last_name ~ '^([a-zA-Za-яА-ЯёЁ]+(?:-[a-zA-Za-яА-ЯёЁ]+)?)$'),
    ADD CONSTRAINT "chk:user.gender:notnull" CHECK (gender IS NOT NULL),
    ADD CONSTRAINT "chk:user.gender:length" CHECK (length(gender) < 32),
    ADD CONSTRAINT "chk:user.gender:regexp" CHECK (
        gender IN ('мужской', 'женский')),
    ADD CONSTRAINT "chk:user.birth_date:notnull" CHECK (
        birth_date IS NOT NULL),
    ADD CONSTRAINT "chk:user.birth_date" CHECK (
        birth_date BETWEEN
            CURRENT_DATE - INTERVAL '120 years' AND
            CURRENT_DATE - INTERVAL '14 years'
    ADD CONSTRAINT "chk:user.role:notnull" CHECK ("role" IS NOT NULL),
    ADD CONSTRAINT "chk:user.role:length" CHECK (length("role") < 32),
    ADD CONSTRAINT "chk:user.role:regexp" CHECK (
        "role" IN ('клиент', 'тренер', 'администратор')
    );
```

Листинг 13 – Реализация ограничений целостности данных отношения Trainer

```
ALTER TABLE "Trainer"
    ADD CONSTRAINT "pk:trainer.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "fk:trainer.user_id" FOREIGN KEY (user_id)
        REFERENCES "User"(id) ON DELETE CASCADE,
    ADD CONSTRAINT "uq:trainer.user_id" UNIQUE (user_id),
    ALTER COLUMN "description" SET DEFAULT 'Нет описания.',
    ADD CONSTRAINT "chk:trainer.description:notnull" CHECK (
        description IS NOT NULL),
    ADD CONSTRAINT "chk:trainer.description:length" CHECK (
        length(description) < 512);
```

Листинг 14 – Реализация ограничений целостности данных отношения TrainerSpecialization

```
ALTER TABLE "TrainerSpecialization"
    ADD CONSTRAINT "pk:trainer_specialization.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "fk:trainer_specialization.trainer_id"
        FOREIGN KEY (trainer_id) REFERENCES "Trainer"(id) ON DELETE CASCADE,
    ADD CONSTRAINT "fk:trainer_specialization.specialization_id"
        FOREIGN KEY (specialization_id)
        REFERENCES "Specialization"(id) ON DELETE CASCADE,
    ADD CONSTRAINT "uq:trainer_specialization.trainer+specialization"
        UNIQUE (trainer_id, specialization_id),
    ALTER COLUMN years SET DEFAULT 0,
    ADD CONSTRAINT "chk:trainer_specialization.years:notnull" CHECK (
        years IS NOT NULL),
    ADD CONSTRAINT "chk:specialization.name:unsigned" CHECK (years >= 0);
```

Листинг 15 – Реализация ограничений целостности данных отношения Training

```
ALTER TABLE "Training"
    ADD CONSTRAINT "pk:training.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "fk:training.specialization_id"
    FOREIGN KEY (specialization_id)
    REFERENCES "Specialization"(id) ON DELETE SET NULL,
    ADD CONSTRAINT "fk:training.room_id" FOREIGN KEY (room_id)
    REFERENCES "TrainingRoom"(id) ON DELETE SET NULL,
    ADD CONSTRAINT "fk:training.trainer_id" FOREIGN KEY (trainer_id)
    REFERENCES "Trainer"(id) ON DELETE SET NULL,
    ALTER COLUMN "date" SET DEFAULT CURRENT_TIMESTAMP,
    ADD CONSTRAINT "chk:training.date:notnull" CHECK ("date" IS NOT NULL);
```

Листинг 16 – Реализация ограничений целостности данных отношения Payment

```
ALTER TABLE "Payment"
    ADD CONSTRAINT "pk:payment.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "fk:payment.membership_id" FOREIGN KEY (membership_id)
    REFERENCES "Membership"(id) ON DELETE SET NULL,
    ALTER COLUMN membership_id SET DEFAULT NULL,
    ADD CONSTRAINT "fk:payment.memshipt_id" FOREIGN KEY (membership_type_id)
    REFERENCES "MembershipType"(id) ON DELETE SET NULL,
    ALTER COLUMN membership_type_id SET DEFAULT NULL,
    ADD CONSTRAINT "fk:payment.user_id" FOREIGN KEY (user_id)
    REFERENCES "User"(id) ON DELETE SET NULL,
    ADD CONSTRAINT "chk:payment.user_id:notnull" CHECK (user_id IS NOT NULL),
    ADD CONSTRAINT "chk:payment.transaction_id:notnull" CHECK (
        transaction_id IS NOT NULL),
    ADD CONSTRAINT "chk:payment.transaction_id:length" CHECK (
        length(transaction_id) < 256),
    ADD CONSTRAINT "uq:payment.transaction_id" UNIQUE (transaction_id),
    ALTER COLUMN "date" SET DEFAULT CURRENT_TIMESTAMP,
    ADD CONSTRAINT "chk:payment.date:notnull" CHECK ("date" IS NOT NULL),
    ALTER COLUMN "method" SET DEFAULT 'наличные',
    ADD CONSTRAINT "chk:payment.method:notnull" CHECK ("method" IS NOT NULL),
    ADD CONSTRAINT "chk:payment.method:length" CHECK (length("method") < 64),
    ADD CONSTRAINT "chk:payment.method:regexp" CHECK (
        "method" IN ('наличные', 'кредитная карта', 'банковский перевод')),
    ALTER COLUMN "gateway" SET DEFAULT NULL,
    ADD CONSTRAINT "chk:payment.gateway:length" CHECK (
        length("gateway") < 64),
    ALTER COLUMN "status" SET DEFAULT 'ожидает',
    ADD CONSTRAINT "chk:payment.status:notnull" CHECK ("status" IS NOT NULL),
    ADD CONSTRAINT "chk:payment.status:length" CHECK (length("status") < 64),
    ADD CONSTRAINT "chk:payment.status" CHECK (
        status IN ('ожидает', 'оплачен', 'отменен'));
```

Листинг 17 – Реализация ограничений целостности данных отношения TrainingRoom

```
ALTER TABLE "TrainingRoom"
    ADD CONSTRAINT "pk:training_room.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "chk:training_room.name:notnull" CHECK (name IS NOT NULL),
    ADD CONSTRAINT "chk:training_room.name:length" CHECK (length(name) < 64),
    ADD CONSTRAINT "uq:training_room.name" UNIQUE (name),
    ADD CONSTRAINT "chk:training_room.capacity:notnull" CHECK (
    capacity IS NOT NULL),
    ADD CONSTRAINT "chk:training_room.capacity" CHECK (capacity > 0);
```

Листинг 18 – Реализация ограничений целостности данных отношения MembershipType

```
ALTER TABLE "MembershipType"
    ADD CONSTRAINT "pk:membership_type.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "chk:membership_type.name:notnull" CHECK (
    name IS NOT NULL),
    ADD CONSTRAINT "chk:membership_type.name:length" CHECK (
    length(name) < 128),
    ADD CONSTRAINT "uq:membership_type.name" UNIQUE (name),
    ALTER COLUMN price SET DEFAULT 0.0,
    ADD CONSTRAINT "chk:membership_type.price:notnull" CHECK (
    price IS NOT NULL),
    ADD CONSTRAINT "chk:membership_type.price:unsigned" CHECK (price >= 0.0),
    ALTER COLUMN sessions SET DEFAULT 1,
    ADD CONSTRAINT "chk:membership_type.sessions:notnull" CHECK (
    sessions IS NOT NULL),
    ADD CONSTRAINT "chk:membership_type.sessions:unsigned" CHECK (
    sessions > 0),
    ALTER COLUMN days SET DEFAULT 1,
    ADD CONSTRAINT "chk:membership_type.days:notnull" CHECK (
    days IS NOT NULL),
    ADD CONSTRAINT "chk:membership_type.days:unsigned" CHECK (days > 0);
```

Листинг 19 – Реализация ограничений целостности данных отношения Membership

```
ALTER TABLE "Membership"
    ADD CONSTRAINT "pk:membership.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "fk:membership.user_id" FOREIGN KEY (user_id)
        REFERENCES "User"(id) ON DELETE CASCADE,
    ADD CONSTRAINT "fk:membership.membership_type_id"
        FOREIGN KEY (membership_type_id)
        REFERENCES "MembershipType"(id) ON DELETE SET NULL,
    ALTER COLUMN start_date SET DEFAULT NULL,
    ALTER COLUMN end_date SET DEFAULT NULL,
    ADD CONSTRAINT "chk:membership.dates:order" CHECK(
        (start_date IS NULL AND end_date IS NULL) OR (start_date IS NOT NULL
        AND end_date IS NOT NULL AND start_date <= end_date)),
    ADD CONSTRAINT "uq:membership.membership_type+user"
        UNIQUE(membership_type_id, user_id),
    ALTER COLUMN available_sessions SET DEFAULT 0,
    ADD CONSTRAINT "chk:membership.available_sessions:notnull" CHECK (
        available_sessions IS NOT NULL),
    ADD CONSTRAINT "chk:membership.available_sessions:unsigned" CHECK (
        available_sessions >= 0);
```

Листинг 20 – Реализация ограничений целостности данных отношения Attendance

```
ALTER TABLE "Attendance"
    ADD CONSTRAINT "pk:attendance.id" PRIMARY KEY (id),
    ALTER COLUMN id SET DEFAULT gen_random_uuid(),
    ADD CONSTRAINT "fk:attendance.membership_id" FOREIGN KEY (membership_id)
        REFERENCES "Membership"(id) ON DELETE CASCADE,
    ADD CONSTRAINT "fk:attendance.training_id" FOREIGN KEY (training_id)
        REFERENCES "Training"(id) ON DELETE SET NULL,
    ADD CONSTRAINT "uq:attendance.membership+training"
        UNIQUE (membership_id, training_id),
    ALTER COLUMN status SET DEFAULT 'ожидает',
    ADD CONSTRAINT "chk:attendance.status:notnull" CHECK (
        status IS NOT NULL),
    ADD CONSTRAINT "chk:attendance.status:length" CHECK (
        length(status) < 64),
    ADD CONSTRAINT "chk:attendance.status:regexp" CHECK (
        status IN ('посетил', 'отсутствовал', 'ожидает'));
```

3.2.1 Реализация триггеров

Реализации триггеров для автоматического обновления количества доступных занятий после посещения тренировки и записи на тренировку с учетом проверки количества доступных мест – представлены на листингах 22 и 21 соответственно.

Листинг 21 – Реализация триггера для записи на тренировку с учетом проверки количества доступных мест

```
CREATE OR REPLACE FUNCTION check_room_capacity()
RETURNS TRIGGER AS $$
BEGIN
    IF (TG_OP = 'INSERT') OR (TG_OP = 'UPDATE' AND NEW.training_id IS
        DISTINCT FROM OLD.training_id)
    THEN
    IF (SELECT COUNT(*) FROM "Attendance" WHERE training_id =
        NEW.training_id) >= (SELECT capacity FROM "TrainingRoom" WHERE id =
        (SELECT room_id  FROM public.training WHERE id = NEW.training_id))
        THEN RAISE EXCEPTION 'Тренировка уже заполнена';
    END IF;
    END IF;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;
CREATE OR REPLACE TRIGGER check_room_capacity_trigger
BEFORE INSERT ON "Attendance" FOR EACH ROW
EXECUTE FUNCTION check_room_capacity();
```

Листинг 22 – Реализация триггера для автоматического обновления количества доступных занятий после посещения тренировки

```
CREATE OR REPLACE FUNCTION update_available_sessions()
RETURNS TRIGGER AS $$
BEGIN
    IF (NEW.status = 'посетил' OR NEW.status = 'отсутствовал')
        AND (OLD.status <> 'посетил' AND OLD.status <> 'отсутствовал')
    THEN
        UPDATE "Membership"
        SET available_sessions = available_sessions - 1
        WHERE id = NEW.membership_id;
    ELSIF (NEW.status = 'ожидает')
        AND (OLD.status = 'посетил' OR OLD.status = 'отсутствовал')
    THEN
        UPDATE "Membership"
        SET available_sessions = available_sessions + 1
        WHERE id = NEW.membership_id;
    END IF;
    RETURN NEW;
END; $$ LANGUAGE plpgsql;
CREATE OR REPLACE TRIGGER attendance_status_update
AFTER UPDATE OF status ON "Attendance"
FOR EACH ROW
EXECUTE FUNCTION update_available_sessions();
```

3.2.2 Создание ролевой модели

Реализация ролевой модели представлена на листингах 23, 24, 25, 26, 27 и 28.

Листинг 23 – Реализация роли Guest (гость)

```
CREATE ROLE guest WITH LOGIN;
GRANT EXECUTE ON FUNCTION register_user(
    UUID, TEXT, TEXT, TEXT, TEXT, TEXT, DATE, TEXT
) TO guest;
GRANT EXECUTE ON FUNCTION login_user(TEXT, TEXT) TO guest;
```

Листинг 24 – Реализация роли Admin (администратор)

```
CREATE ROLE admin WITH LOGIN PASSWORD 'admin' SUPERUSER BYPASSRLS;
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO admin;
```

Листинг 25 – Реализация ролей Client (клиент) и Trainer (тренер) – начало

```
CREATE ROLE client WITH LOGIN PASSWORD 'client';
CREATE ROLE trainer WITH LOGIN PASSWORD 'trainer';
```

Листинг 26 – Реализация ролей Client (клиент) и Trainer (тренер) – продолжение

```
GRANT client TO trainer;
GRANT ALL ON "User" TO client;
GRANT ALL ON "User" TO trainer;
ALTER TABLE "User" ENABLE ROW LEVEL SECURITY;
CREATE POLICY personal_client_trainer_user_policy
    ON "User"
    FOR ALL
    TO client, trainer
    USING (id = current_user_id());
CREATE POLICY public_client_trainer_user_policy
    ON "User"
    FOR SELECT
    TO client, trainer
    USING (TRUE);
GRANT ALL ON "Trainer" TO client;
GRANT ALL ON "Trainer" TO trainer;
ALTER TABLE "Trainer" ENABLE ROW LEVEL SECURITY;
CREATE POLICY public_client_trainer_trainer_policy
    ON "Trainer"
    FOR SELECT
    TO client, trainer
    USING (TRUE);
CREATE POLICY personal_trainer_trainer_policy
    ON "Trainer"
    FOR ALL
    TO trainer
    USING (user_id = current_user_id());
GRANT ALL ON "Membership" TO client;
GRANT ALL ON "Membership" TO trainer;
ALTER TABLE "Membership" ENABLE ROW LEVEL SECURITY;
CREATE POLICY client_trainer_membership_policy
    ON "Membership"
    FOR ALL
    TO client, trainer
    USING (user_id = current_user_id());
CREATE POLICY trainer_membership_policy
    ON "Membership"
    FOR SELECT
    TO trainer
    USING (TRUE);
GRANT ALL ON "Payment" TO client;
GRANT ALL ON "Payment" TO trainer;
ALTER TABLE "Payment" ENABLE ROW LEVEL SECURITY;
```

Листинг 27 – Реализация ролей Client (клиент) и Trainer (тренер) – продолжение

```
CREATE POLICY client_trainer_payment_policy
    ON "Payment"
    FOR ALL
    TO client, trainer
    USING (user_id = current_user_id());
GRANT ALL ON "Attendance" TO client;
GRANT ALL ON "Attendance" TO trainer;
ALTER TABLE "Attendance" ENABLE ROW LEVEL SECURITY;
CREATE POLICY client_trainer_attendance_policy
    ON "Attendance"
    FOR ALL
    TO client, trainer
    USING (
        membership_id IN (
            SELECT id
            FROM "Membership"
            WHERE user_id = current_user_id()
        )
    );
CREATE POLICY trainer_attendance_policy
    ON "Attendance"
    FOR ALL
    TO trainer
    USING (
        training_id IN (
            SELECT id
            FROM "Training"
            WHERE trainer_id = (
                SELECT id
                FROM "Trainer"
                WHERE user_id = current_user_id()
            )
        )
    );
GRANT SELECT ON "Specialization" TO client;
GRANT SELECT ON "Specialization" TO trainer;
GRANT SELECT ON "TrainerSpecialization" TO client;
GRANT SELECT ON "TrainerSpecialization" TO trainer;
GRANT ALL ON "Training" TO trainer;
GRANT ALL ON "Training" TO client;
ALTER TABLE "Training" ENABLE ROW LEVEL SECURITY;
CREATE POLICY client_training_select_policy
    ON "Training"
    FOR SELECT
```

Листинг 28 – Реализация ролей Client (клиент) и Trainer (тренер) – конец

```
    TO client, trainer
    USING (TRUE);
CREATE POLICY trainer_training_update_policy
    ON "Training"
    FOR ALL
    TO trainer
    USING (
        trainer_id = (
            SELECT id
            FROM "Trainer"
            WHERE user_id = current_user_id()
        )
    );
GRANT SELECT ON "MembershipType" TO client;
GRANT SELECT ON "MembershipType" TO trainer;
GRANT SELECT ON "TrainingRoom" TO client;
GRANT SELECT ON "TrainingRoom" TO trainer;
```

3.3 Интерфейс для взаимодействая с базой данных

Для взаимодействия с базой данных было разработано серверное программное обеспечение. Серверная часть предоставляет набор точек доступа по протоколу HTTP [13] для выполнения операций создания, чтения, обновления и удаления данных. Все запросы обрабатываются в соответствии с принципами REST [14] и документированы с использованием инструмента Swagger [15], что обеспечивает удобный доступ к спецификации прикладного программного интерфейса через веб-интерфейс (см. таблицу 14).

На рисунках 11 и 12 показан интерфейс Swagger, который предоставляет визуализацию документации REST API и позволяет выполнять интерактивные запросы к сервису.

Метод	**Путь**	**Описание**
POST	/admin/attendances	Создать новое посещение
GET	/admin/attendances/all	Получить список всех посещений
DELETE	/admin/attendances/{id}	Удалить посещение по ID
PUT	/admin/attendances/{id}	Обновить посещение по ID

Таблица 14 – Примеры операций интерфейса доступа

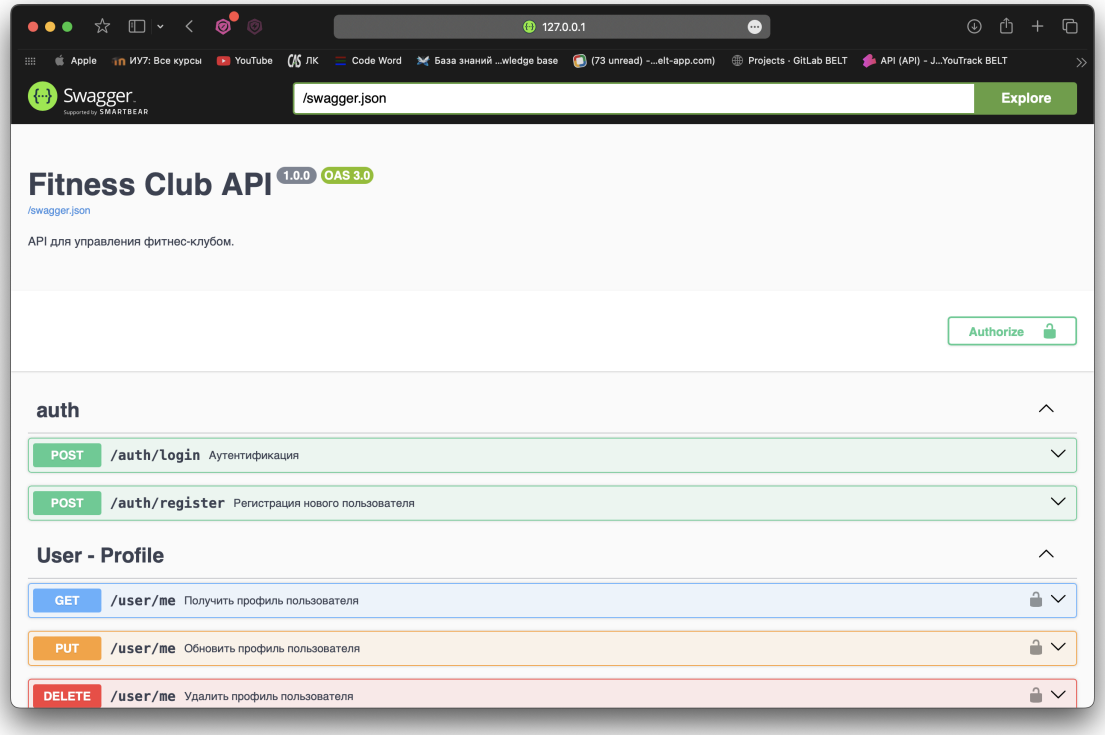

Рисунок 11 – Главная страница интерфейса Swagger с перечнем доступных API

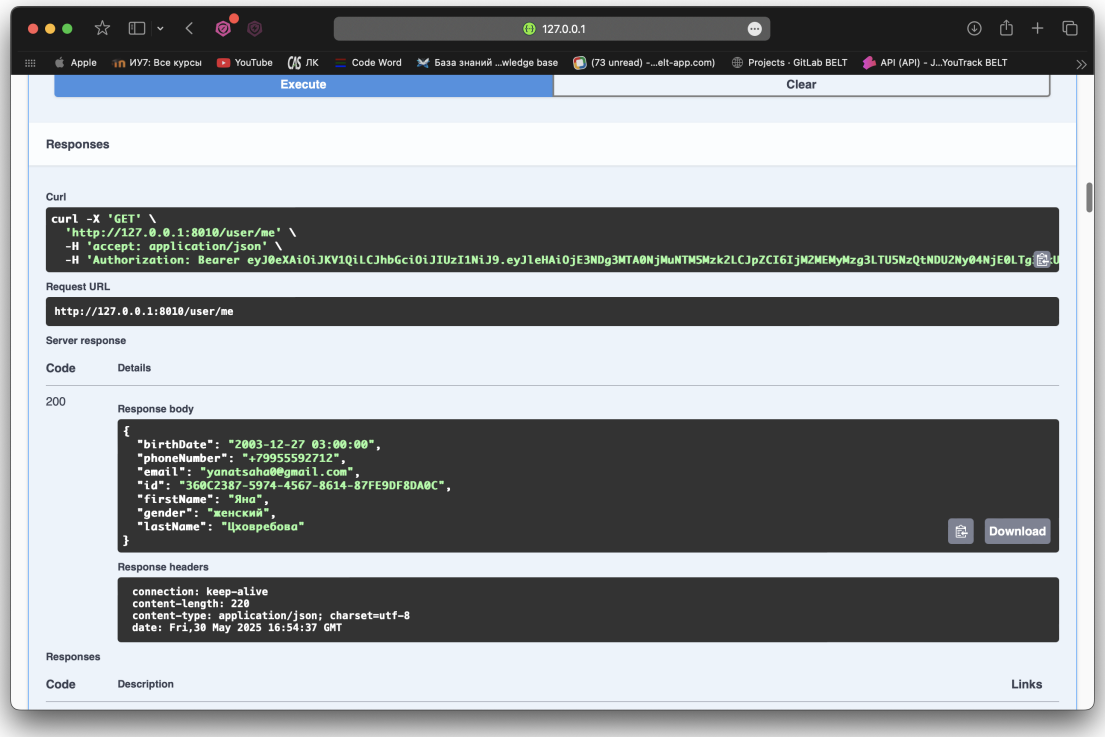

Рисунок 12 – Пример запроса к эндпоинту *GET /user/me* для получения данных авторизованного пользователя

3.4 Тестирование

В таблицах 15 и 16 приведены результаты тестирования триггеров для автоматического обновления количества доступных занятий после посещения тренировки и для записи на тренировку с учётом проверки доступных мест.

Таблица 15 – Проверка корректности работы триггеров PostgreSQL – начало

№	Входные данные	Ожидаемый результат	Фактический результат
Триггер update_available_sessions			
6	Статус *Attendance* меняется с «ожидает» на «посетил»	Значение *available_sessions* уменьшается на 1	Уменьшено на 1
7	Статус *Attendance* меняется с «ожидает» на «отсутствовал»	Значение *available_sessions* уменьшается на 1	Уменьшено на 1
8	Статус *Attendance* меняется с «посетил» на «ожидает»	Значение *available_sessions* увеличивается на 1	Увеличено на 1
9	Статус*Attendance* меняется с «отсутствовал» на «ожидает»	Значение *available_sessions* увеличивается на 1	Увеличено на 1
10	Статус*Attendance* меняется с «посетил» на «посетил»	*available_sessions* не изменяется	Значение не изменилось
11	Статус *Attendance* меняется с «ожидает» на «ожидает»	*available_sessions* не изменяется	Значение не изменилось
12	Запись *Attendance* изменяется, но статус не трогается	*available_sessions* не изменяется	Значение не изменилось

Таблица 16 – Проверка корректности работы триггеров PostgreSQL – конец

№	Входные данные	Ожидаемый результат	Фактический результат
Триггер check_room_capacity			
1	Вставка новой записи в *Attendance*, *training_id* указывает на тренировку с незаполненной комнатой	Запись добавляется успешно	Запись добавлена
2	Вставка новой записи в *Attendance*, когда количество уже достигло *capacity* в *TrainingRoom*	Ошибка: *Тренировка уже заполнена*	Ошибка выдана

Вывод

В данном разделе представлены выбранные средства реализации, приведены реализации сущностей, ограничений целостности данных, ролевой модели и триггеров. Также описано тестирование триггеров и описан интерфейс, обеспечивающий взаимодействие с базой данных.

4 Исследовательская часть

В данном разделе представлены результаты исследований, проведённых для оценки производительности системы при увеличении объёма данных, анализа скорости получения ответа в зависимости от количества одновременных запросов, а также сравнения скорости обработки с использованием кэширования и без него.

4.1 Технические характеристики

Исследования проводилось на устройстве со следующими техническими характеристиками:

— операционная система macOS 14.6.1 [16];

— оперативная память (RAM) 16 ГБ;

— процессор (CPU) 2 ГГц 4-ядерный Intel Core i5 (8 логических ядер) [17].

Во время проведения исследования, устройство было подключено к сети электропитания и не было нагружено сторонними приложениями, за исключением встроенных приложений окружения.

Влияние объема данных на время выполнения

В таблице 17 представлены усреднённые значения исследования времени выполнения операций INSERT, SELECT, UPDATE и DELETE в зависимости от количества строк в таблице User. Измерения проводились при объёмах данных от 100 до 99,1000 записей с шагом 1,000. Для каждой операции выполнялось 10 повторений, после чего вычислялось среднее время выполнения одной операции в миллисекундах.

На рисунке 13 приведена диаграмма, иллюстрирующая зависимость времени выполнения каждой операции от размера таблицы.

В приложении А приведены листинги кода, использованные при проведении исследования производительности операций INSERT, SELECT, UPDATE и DELETE на таблице User.

Таблица 17 – Среднее время операций с таблицей User (в миллисекундах)

Размер таблицы	INSERT	SELECT	UPDATE	DELETE
100	0.0166	0.0325	0.0396	0.0307
1100	0.0183	0.2800	0.1231	0.0951
5100	0.0332	1.1239	0.5739	0.0871
10100	0.0449	2.6399	1.1599	0.9866
25100	0.0670	11.6802	6.0323	5.2592
50100	0.0637	14.2658	5.6018	6.1221
99100	0.0758	23.2923	9.3653	9.3532

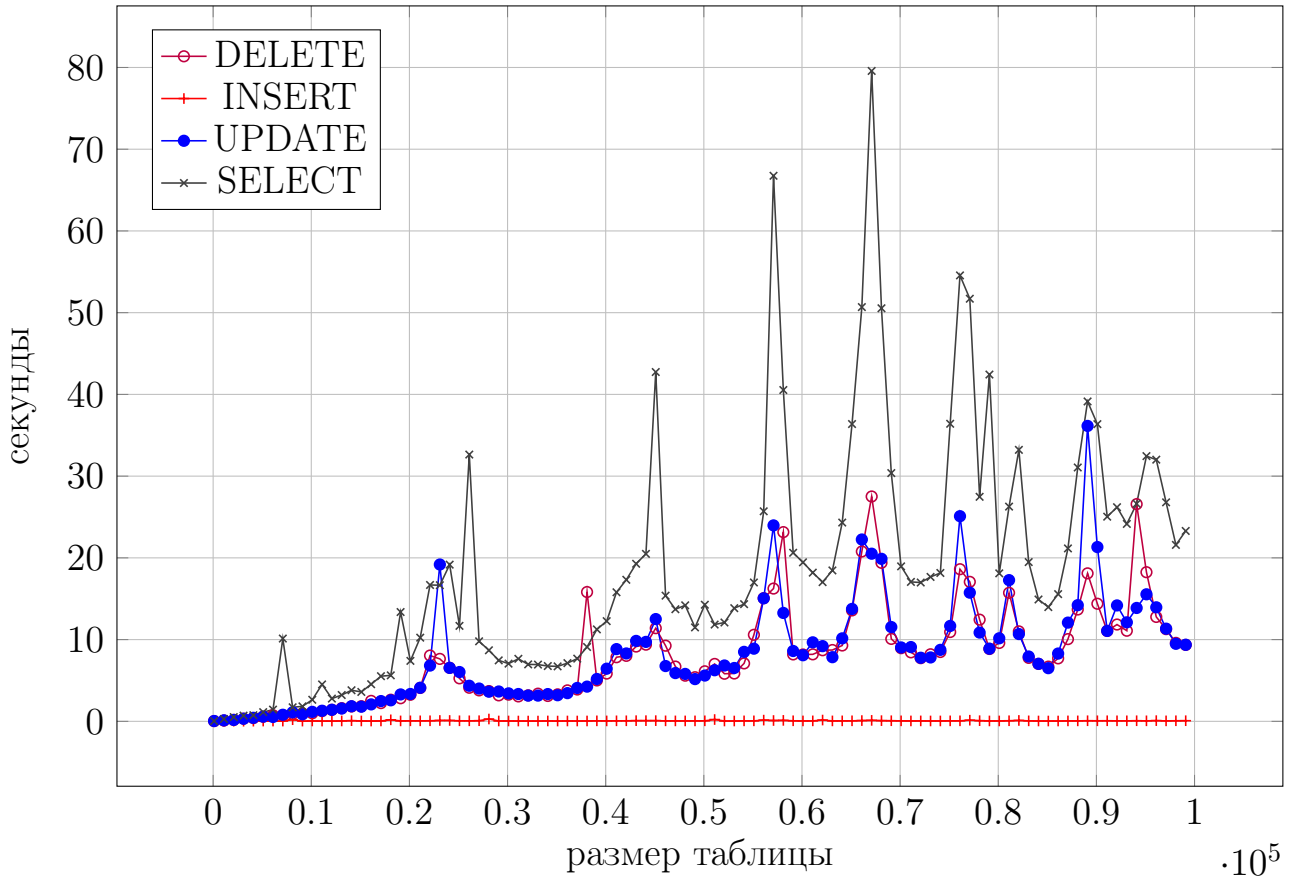

Рисунок 13 – Зависимость времени выполнения операций от записей в таблице

Вывод

Полученные результаты (в миллисекундах) показали следующее:

— **INSERT** – время выполнения увеличилось с 0.0167 мс при 100 записях до 0.0758 мс при 99 100 записях – рост на $\approx 354\%$;

— **SELECT** – время выполнения увеличилось с 0.0325 мс до 23.2923 мс – рост более чем на 71 500%;

— **UPDATE** – время выполнения увеличилось с 0.0396 мс до 9.3653 мс – рост примерно на 23 600%;

— **DELETE** – время выполнения увеличилось с 0.0307 мс до 9.3532 мс – рост на 30 400%.

Таким образом, при увеличении объёма данных операции *SELECT*, *UPDATE* и *DELETE* оказываются примерно в одинаковой мере медленнее по сравнению с *INSERT*, который сохраняет относительную быстроту даже при больших объёмах.

В результате серии измерений были выявлены единичные «пики» задержек при выполнении операций. Такие выбросы обусловлены не ростом сложности самого запроса, а фоновой активностью СУБД и операционной системы.

Влияние количества одновременных запросов на отклик

В таблице 18 приведены результаты исследования времени отклика от количества клиентов в миллисекундах в течение 10 секунд. На рисунке 14 изображено графическое представление данных из таблицы 18.

На листинге 29 приведён SQL-запрос, используемый для измерения времени отклика.

Таблица 18 – Результаты нагрузочного исследования

Количество клиентов	**Среднее время отклика (мс)**	**Количество запросов**
1	0.272817	36 550
10	1.516456	65 898
25	5.964658	41 914
50	11.333167	44 124
75	13.321161	56 332

Листинг 29 – Извлечение данных о пользователях мужского пола

```
SELECT *
FROM "User"
WHERE gender = 'мужской';
```

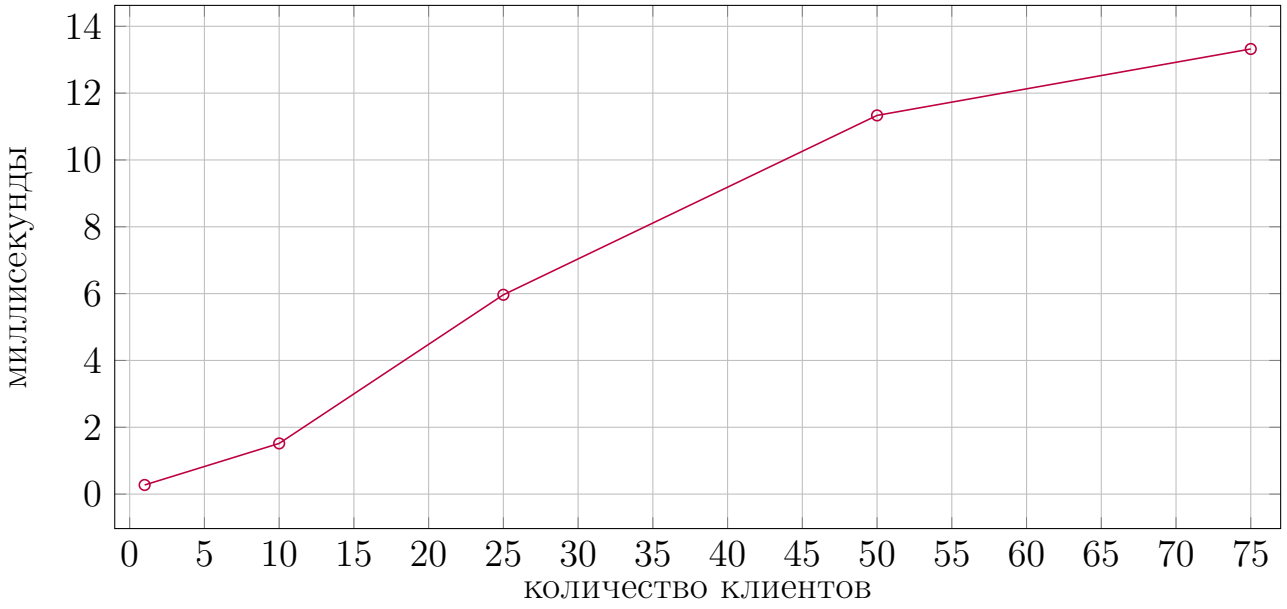

Рисунок 14 – Зависимость времени отклика от количества клиентов

Вывод

Результаты второго исследования показали, что с увеличением количества клиентов время отклика на запросы также увеличивается. Так, при 10 клиентах среднее время отклика составило 1.5165 мс, а при 75 клиентах – 13.3212 мс. Это свидетельствует о том, что с ростом числа одновременных запросов наблюдается значительное увеличение времени отклика, что указывает на увеличение нагрузки на базу данных.

Влияние кэширования на скорость запросов

В рамках исследования была оценена производительность запросов к данным, получаемым из базы данных PostgreSQL и Redis-кэша.

В данном исследовании для оценки скорости выборки из PostgreSQL и Redis-кэша использовался SQL-запрос, представленный на листинге 30.

Листинг 30 – Извлечение данных о пользователях женского пола, рождённых после 1 января 2003 г., с сортировкой по дате рождения

```
SELECT * FROM "User" u
JOIN "Membership" m on m.user_id = u.id
JOIN "Order" o on o.id = m.order_id
WHERE birth_date > '2003-01-01' AND gender = 'женский'
ORDER BY birth_date DESC;
```

Для каждого из источников данных были замерены времена отклика на запросы. Каждый запрос к кэшу и базе данных выполнялся 30 раз. Исследование проводилось на протяжении нескольких минут, с интервалом в 5 секунд между запросами. Кэш в Redis хранился в течение 10 секунд.

Среднее время выполнения запросов к Redis-кэшу (Кэш) составило 0.00188 секунды, для базы данных PostgreSQL (БД) – 0.00246 секунды. На рисунке 15 изображено графическое представление полученных данных.

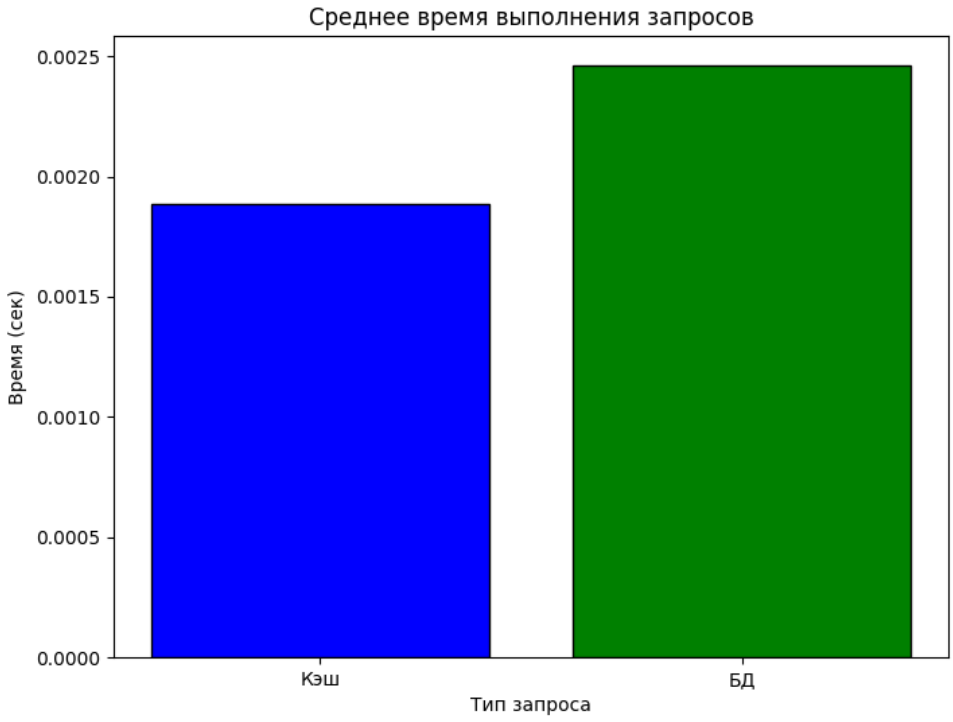

Рисунок 15 – Сравнение времени выполнения запросов к кэшу и базе данных

Вывод

Результаты третьего исследования показали, что кэширование ускоряет выполнение запросов. Среднее время выполнения запроса к Redis-кэшу составило 0.00188 секунды, что на 0.00058 секунды быстрее, чем запросы к базе данных PostgreSQL, где среднее время составило 0.00246 секунды. Это подтверждает эффективность использования кэширования для ускорения обработки данных.

ЗАКЛЮЧЕНИЕ

Цель курсовой работы, заключавшаяся в разработке базы данных для хранения и обработки данных фитнес-клуба, была успешно достигнута. В ходе работы были выполнены все поставленные задачи.

Был проведен анализ предметной области и формализована задача, что позволило определить требования к базе данных. Разработана структура базы данных с определением ролей пользователей системы. После анализа различных моделей данных была выбрана наиболее оптимальная модель для данного проекта.

В процессе проектирования базы данных были спроектированы необходимые сущности и их взаимосвязи, а также ролевая модель для разграничения прав доступа. Также были разработаны триггеры для автоматического обновления данных и внедрены ограничения целостности данных, что обеспечило корректность хранимой информации. Все проектные решения были успешно реализованы, а триггеры протестированы.

Для взаимодействия с базой данных был реализован интерфейс.

Проведенные исследования производительности базы данных показали, что кеширование значительно улучшает время отклика по сравнению с прямыми запросами к базе данных. Также было установлено, что с увеличением объема данных и количества одновременных запросов время отклика системы увеличивается, что указывает на рост нагрузки на базу данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. IHRSA. The 2019 IHRSA Global Report. – Boston, MA, USA: Racquet and Sportsclub Associatio, 2019.

2. León-Quismondo J. Service Perceptions in Fitness Centers: IPA Approach by Gender and Age / León-Quismondo J., García-Unanue J., and Burillo P. // International Journal of Environmental Research and Public Health. 2020. Vol. 17, no. 8. p. 2844.

3. Bates M. Health Fitness Management: A Comprehensive Resource for Managing and Operating Programs and Facilities. 2nd edition. – Champaign, IL, USA: Human Kinetics, 2019.

4. Аврунев О. Е. Стасышин В. М. Модели баз данных. – Новосибирск: Новосибирский государственный технический университет, 2018.

5. Рейтинг систем управления базами данных (DB-Engines). Режим доступа: https://db-engines.com/en/ranking (дата обращения: 25.03.2025).

6. Codd E. F. A Relational Model of Data for Large Shared Data Banks // Communications of the ACM. 1970. Vol. 13, no. 6. P. 377–387.

7. Gray J. The Transaction Concept: Virtues and Limitations. – Cupertino, CA, USA: Tandem Computers Incorporated, 1981.

8. PostgreSQL Global Development Group. PostgreSQL: The world's most advanced open source database. Режим доступа: https://www.postgresql.org/ (дата обращения: 27.03.2025).

9. Sanfilippo Salvatore. Redis: A persistent key-value database. Режим доступа: https://redis.io/ (дата обращения: 27.03.2025).

10. Apple Inc. Swift Programming Language. Режим доступа: https://swift.org (дата обращения: 27.03.2025).

11. Apple Inc. Xcode. Режим доступа: https://developer.apple.com/xcode/ (дата обращения: 26.03.2025). 2025.

12. pgAdmin Development Team. pgAdmin 4. Режим доступа: https://www.pgadmin.org/ (дата обращения: 26.03.2025).

13. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Режим доступа: https://tools.ietf.org/html/rfc7230 (дата обращения: 03.04.2024).

14. Fielding Roy T. Architectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis. – Irvine, CA, USA: University of California, 2000.

15. Swagger — OpenAPI Specification. Режим доступа: https://swagger.io/specification/ (дата обращения: 03.04.2024).

16. Apple Inc. macOS 14 Sonoma. Режим доступа: https://www.apple.com/macos/sonoma/ (дата обращения: 09.04.2025).

17. Документация процессора Intel Core i5-10210U [Электронный ресурс]. Режим доступа: https://ark.intel.com/content/www/us/en/ark/products/195436/intel-core-i510210u-processor-6m-cache-up-to-4-20-ghz.html (Дата обращения: 17.10.2024).

ПРИЛОЖЕНИЕ А

Листинг 31 – Функция генерации тестовых пользователей

```
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE OR REPLACE FUNCTION generate_test_users(num_records INT)
RETURNS VOID AS $$
BEGIN
    INSERT INTO "User" (
        id, email, phone_number, password,
        first_name, last_name, gender, birth_date, role
    )
    SELECT
        gen_random_uuid(),
        'user' || i || '@example.com',
        '+7' || lpad((i + 1000000000)::text, 10, '0'),
        'password' || i,
        initcap(substr(translate(md5(random()::text),
            '0123456789','abcdefghij'),1,8)),
        initcap(substr(translate(md5(random()::text),
            '0123456789','klmnopqrst'),1,10)),
        CASE WHEN random()>0.5 THEN 'мужской' ELSE 'женский' END,
        CURRENT_DATE - (365*20 + floor(random()*365*55))::int,
        'клиент'
    FROM generate_series(1, num_records) i;
END;
$$ LANGUAGE plpgsql;
```

Листинг 32 – Таблица для хранения результатов замеров времени

```
DROP TABLE IF EXISTS performance_metrics;
CREATE TABLE performance_metrics (
    table_size   INT,
    operation    TEXT,
    avg_time_ms  DOUBLE PRECISION
);
```

Листинг 33 – Исследование производительности операций INSERT, SELECT, UPDATE и DELETE на таблице User – начало

```
DO $$
DECLARE
    sizes int[] := ARRAY(SELECT generate_series(100, 100000,
        1000));
    sz int;
    total_time double precision;
```

Листинг 34 – Исследование производительности операций INSERT, SELECT, UPDATE и DELETE на таблице User – продолжение

```
    avg_time double precision;
    i int;
    u_id UUID;
    t1 timestamp;
    t2 timestamp;
BEGIN
    FOREACH sz IN ARRAY sizes LOOP
        RAISE NOTICE '========== % записей ========== ', sz;

        -- INSERT тест
        TRUNCATE TABLE "User" RESTART IDENTITY CASCADE;
        PERFORM generate_test_users(sz);
        total_time := 0;
        FOR i IN 1..10 LOOP
            u_id := gen_random_uuid();
            t1 := clock_timestamp();
            INSERT INTO "User"(
                id, email, phone_number, password,
                first_name, last_name, gender, birth_date, role
            )
            VALUES (
                u_id,
                'test' || i || '@example.com',
                '+79001234567',
                'pass',
                'Имя',
                'Фамилия',
                'мужской',
                CURRENT_DATE - INTERVAL '30 years',
                'клиент'
            );
            t2 := clock_timestamp();
            DELETE FROM "User" WHERE id = u_id;
            total_time := total_time + EXTRACT(EPOCH FROM t2 -
                t1) * 1000;
        END LOOP;
        avg_time := total_time / 10;
        INSERT INTO performance_metrics VALUES (sz, 'INSERT',
            avg_time);
```

Листинг 35 – Исследование производительности операций INSERT, SELECT, UPDATE и DELETE на таблице User – продолжение

```
        -- SELECT тест
        TRUNCATE TABLE "User" RESTART IDENTITY CASCADE;
        PERFORM generate_test_users(sz);
        total_time := 0;
        FOR i IN 1..10 LOOP
            t1 := clock_timestamp();
            SELECT id INTO u_id FROM "User" ORDER BY random()
                LIMIT 1;
            t2 := clock_timestamp();
            PERFORM 1 FROM "User" WHERE id = u_id;
            total_time := total_time + EXTRACT(EPOCH FROM t2 -
                t1) * 1000;
        END LOOP;
        avg_time := total_time / 10;
        INSERT INTO performance_metrics VALUES (sz, 'SELECT',
            avg_time);

        -- UPDATE тест
        TRUNCATE TABLE "User" RESTART IDENTITY CASCADE;
        PERFORM generate_test_users(sz);
        total_time := 0;
        FOR i IN 1..10 LOOP
            SELECT id INTO u_id FROM "User" ORDER BY random()
                LIMIT 1;
            t1 := clock_timestamp();
            UPDATE "User" SET first_name = 'Обновлено' WHERE id =
                u_id;
            t2 := clock_timestamp();
            total_time := total_time + EXTRACT(EPOCH FROM t2 -
                t1) * 1000;
        END LOOP;
        avg_time := total_time / 10;
        INSERT INTO performance_metrics VALUES (sz, 'UPDATE',
            avg_time);
```

Листинг 36 – Исследование производительности операций INSERT, SELECT, UPDATE и DELETE на таблице User – конец

```
        -- DELETE тест
        total_time := 0;
        FOR i IN 1..10 LOOP
            SELECT id INTO u_id FROM "User" ORDER BY random()
                LIMIT 1;
            t1 := clock_timestamp();
            DELETE FROM "User" WHERE id = u_id;
            t2 := clock_timestamp();
            INSERT INTO "User"(
                id, email, phone_number, password,
                first_name, last_name, gender, birth_date, role
            )
            VALUES (
                gen_random_uuid(),
                'repl'||i||'@example.com',
                '+79990000000',
                'pass',
                'Имя',
                'Фамилия',
                'женский',
                CURRENT_DATE - INTERVAL '25 years',
                'клиент'
            );
            total_time := total_time + EXTRACT(EPOCH FROM t2 -
                t1) * 1000;
        END LOOP;
        avg_time := total_time / 10;
        INSERT INTO performance_metrics VALUES (sz, 'DELETE',
            avg_time);
        TRUNCATE TABLE "User" RESTART IDENTITY CASCADE;
    END LOOP;
END $$;
```

ПРИЛОЖЕНИЕ Б

Презентация состоит из 17 слайдов.