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

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«Разработка базы данных для хранения информации о репетиционных базах»

Студент ИУ7-66Б _____ **А. А. Петрова**
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель _____ **М. В. Филиппов**
(Подпись, дата) (И.О. Фамилия)

2022 г.

РЕФЕРАТ

Расчетно-пояснительная записка 36 с., 10 рис., 5 табл., 7 ист., 5 прил.

Объектом разработки в данной работе является база данных, содержащая информацию о репетиционных базах, соответствующих им комнатах и оборудовании, с целью предоставить возможность пользователям искать необходимые комнаты и бронировать свои репетиции. Цель данной работы – реализовать приложение, содержащее информацию о репетиционных базах. В приложении, работающем с этой БД, должна быть возможность для музыканта бронировать или отменять свои репетиции, а для владельца реп. базы - отслеживать записи на свою реп. базу.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- формализовать задание, определить необходимый функционал;
- провести анализ СУБД;
- описать структуру БД;
- создать и заполнить БД;
- разработать ПО, которое позволит пользователю-музыканту бронировать и отменять свои репетиции, а владельцу отслеживать их;
- провести исследование зависимости времени выполнения запроса от числа записей в таблице.

Поставленная цель достигнута: в ходе курсового проекта была разработана база данных, хранящая информацию о репетиционных точках. При этом при разработке в качестве СУБД использовался PostgreSQL, а в качестве языка программирования – Python 3.7.

Дальнейшее развитие проекта подразумевает:

- добавление фотографий комнат в блок информации о них;
- добавление календаря для более удобного бронирования репетиций;
- добавление возможности бронировать не только репетиционные базы, но

и другие творческие площадки;

- добавление возможности администраторам блокировать пользователей;
- создание мобильной версии приложения.

КЛЮЧЕВЫЕ СЛОВА

базы данных, разработка ПО, репетиционные базы, бронирование репетиций, postgresql, python.

СОДЕРЖАНИЕ

Введение

Проблема поиска места для репетиций является актуальной для любого музыканта, тем более группы. В крупных городах есть достаточно много репетиционных баз (далее: реп. баз). Все они имеют разные цены и характеристики. Поэтому существует потребность в приложении, которое собирало бы воедино всю имеющуюся информацию о различных репетиционных базах, таким образом освобождая музыкантов от необходимости вручную искать и изучать каждую реп. базу, заходить на их сайты, звонить лично, чтобы забронировать репетицию и т. д.

Цель данной работы – реализовать приложение, содержащее информацию о репетиционных базах. В приложении, работающем с этой БД, должна быть возможность для музыканта бронировать или отменять свои репетиции, а для владельца реп. базы - отслеживать записи на свою реп. базу.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- формализовать задание, определить необходимый функционал;
- провести анализ СУБД;
- описать структуру БД;
- создать и заполнить БД;
- разработать ПО, которое позволит пользователю-музыканту бронировать и отменять свои репетиции, а владельцу отслеживать их;
- провести исследование зависимости времени выполнения запроса от числа записей в таблице.

1 Аналитическая часть

1.1 Анализ задачи и существующих решений

Необходимо разработать программу для отображения информации о репетиционных базах с возможностью для музыканта бронировать или отменять свои репетиции, а для владельца реп. базы - отслеживать записи на свою реп. базу.

На сегодняшний день существует всего два подобных приложения:

- MUSbooking

  Это наиболее известное и популярное приложение по бронированию творческих площадок.

  Его основные преимущества: возможность бронирования любых творческих площадок и возможность бронирования в других городах России (не только в Москве).

  Основной недостаток: нельзя посмотреть сразу весь список зарегистрированных реп. баз в текущем городе. Список доступных реп. баз можно посмотреть только после указания конкретного времени репетиции, что крайне неудобно в определённых ситуациях.

- TONESKY

  Основное преимущество: возможность заранее посмотреть весь список зарегистрированных реп. баз.

  Из недостатков: отсутствие поиска по реп. базам, отсутствие цены на превью (т. е., чтобы посмотреть цену, надо зайти «вглубь» и посмотреть подробную информацию), ориентация только на Москву (не существенно в рамках моей задачи).

1.2 Формализация данных

База данных должна хранить информацию о:

- репетиционной базе и её комнатах;
- оборудовании;

- пользователях (музыкантах и владельцах) и об их забронированных репетициях или реп. базах соответственно.

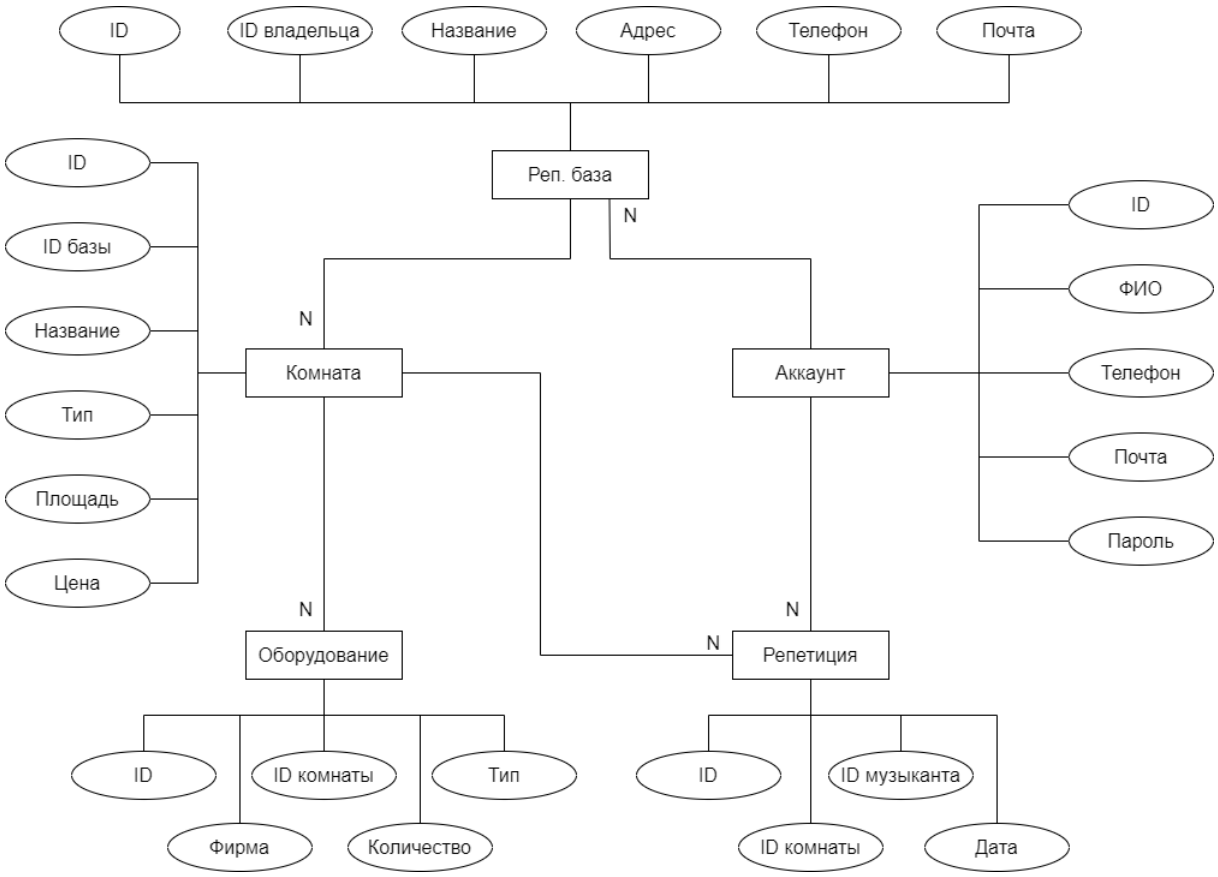

Рисунок 1 – ER-диаграмма разрабатываемой базы данных

Таблица 1 – Категории и сведения о данных

Категория	**Сведения**
Реп. база	Название, адрес, телефон, почта, кому принадлежит
Комната	Название, тип (вокал/группа и т. д.), площадь, стоимость за 3 часа, к какой репбазе относится
Оборудование в комнате	Тип (усилитель/ударные/микрофон и т. д.), бренд, количество, к какой комнате относится
Аккаунт	ФИО, телефон, почта
Репетиция	Время, какой музыкант (аккаунт), какая комната

1.3 Типы пользователей

Из задачи ясно, что есть два типа пользователей: обычный музыкант и владелец реп. базы. Помимо этого, будет также выделена роль администратора приложения. Для всех троих будет нужна авторизация.

Таблица 2 – Типы пользователей и их полномочия

Тип пользователя	**Полномочия**
Музыкант	Бронирование репетиций, отмена репетиций, просмотр брони
Владелец	Добавление репбазы, удаление своей репбазы, просмотр записей на репетиции, отмена репетиций
Администратор	Удаление репбазы, отмена репетиций, просмотр бронирований

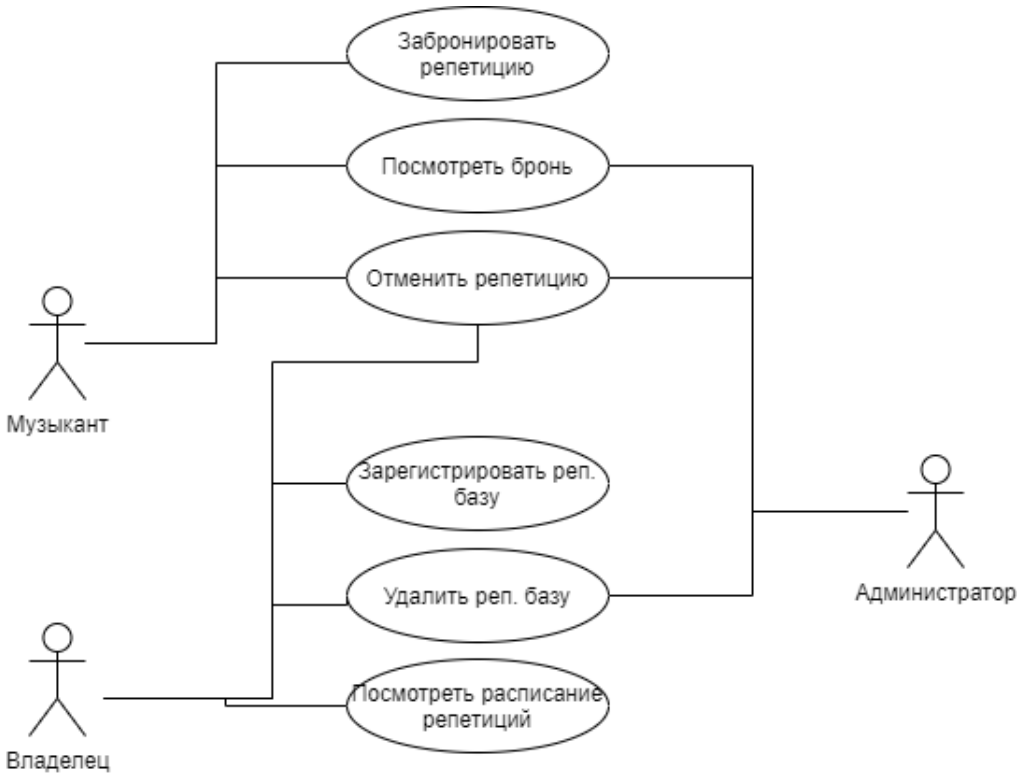

Рисунок 2 – Use-case диаграмма приложения

1.4 Классификации СУБД

Система управления базами данных (СУБД) – совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных [1].

Основными функциями СУБД являются:

- управление данными на внешней памяти;
- управление данными в оперативной памяти с использованием дискового кэша;
- журнализация изменений, резервное копирование и восстановление базы данных после сбоев;
- поддержка языков БД.

Классификации СУБД:

По модели данных:

- дореляционные;

Иерархические. Это модель данных, где используется представление базы данных в виде древовидной (иерархической) структуры, состоящей из объектов (данных) различных уровней.

Сетевые. Это логическая модель данных, являющаяся расширением иерархического подхода, строгая математическая теория, описывающая структурный аспект, аспект целостности и аспект обработки данных в сетевых базах данных.

Разница между иерархической моделью данных и сетевой состоит в том, что в иерархических структурах запись-потомок должна иметь в точности одного предка, а в сетевой структуре данных у потомка может иметься любое число предков.

- реляционные.

  Реляционная модель данных является совокупностью данных и состоит из набора двумерных таблиц. При табличной организации отсутствует иерархия элементов. Таблицы состоят из строк – записей и столбцов – полей. На пересечении строк и столбцов находятся конкретные значения. Для каждого поля определяется множество его значений. За счет возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов.

  Реляционная модель является удобной и наиболее широко используемой формой представления данных.

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных [2].

По степени распределённости:

- локальные (все части локальной СУБД размещаются на одном компьютере);

- распределённые (части СУБД могут размещаться не только на одном, но на двух и более компьютерах).

По способу доступа к БД:

- файл-серверные;

В файл-серверных СУБД файлы данных располагаются централизованно на файл-сервере. СУБД располагается на каждом клиентском компьютере (рабочей станции). Доступ СУБД к данным осуществляется через локальную сеть. Синхронизация чтений и обновлений осуществляется посредством файловых блокировок.

Преимуществом этой архитектуры является низкая нагрузка на процессор файлового сервера.

Недостатки: потенциально высокая загрузка локальной сети; затруднённость или невозможность централизованного управления; затруднённость или невозможность обеспечения таких важных характеристик, как высокая надёжность, высокая доступность и высокая безопасность. Применяются чаще всего в локальных приложениях, которые используют функции управления БД; в системах с низкой интенсивностью обработки данных и низкими пиковыми нагрузками на БД.

На данный момент файл-серверная технология считается устаревшей, а её использование в крупных информационных системах — недостатком [3].

Пример: Microsoft Access.

- клиент-серверные;

Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно.

Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу.

Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик, как высокая надёжность, высокая доступность и высокая безопасность.

Примеры: Oracle Database, MS SQL Server, PostgreSQL, MySQL.

- встраиваемые.

  СУБД, которая может поставляться как составная часть некоторого программного продукта, не требуя процедуры самостоятельной установки. Встраиваемая СУБД предназначена для локального хранения данных своего приложения и не рассчитана на коллективное использование в сети.

  Физически встраиваемая СУБД чаще всего реализована в виде подключаемой библиотеки. Доступ к данным со стороны приложения может происходить через SQL либо через специальные программные интерфейсы.

  Примеры: SQLite, Microsoft SQL Server Compact.

Выводы

В этом разделе была проанализирована поставленная задача и уже существующие решения. А также были формализованы данные, типы пользователей и их полномочия. После чего были рассмотрены разные типы СУБД и их функции.

2 Конструкторская часть

2.1 Проектирование БД

База данных должна хранить представленные в таблице 1 данные. Исходя из этого, в проектируемой базе данных можно выделить следующие таблицы:

- таблица с реп. базами (rehbase);
- таблица с комнатами реп. баз (room);
- таблица с оборудованием в комнатах (equipment);
- таблица с аккаунтами пользователей (account);
- таблица с забронированными репетициями (rehearsal).

Таблица rehbase должна содержать информацию о:

- id – идентификатор реп. базы (PK);
- name – название реп. базы (text);
- address – адрес реп. базы (text);
- phone – номер телефона для связи (text);
- mail – электронная почта для связи (text);
- ownerid – id владельца реп. базы (связь с таблицей account) (FK, один-ко-многим).

Таблица room должна содержать информацию о:

- id – идентификатор комнаты (PK);
- name – название комнаты (text);
- type – тип комнаты (text);
- area – площадь комнаты (integer);
- cost – стоимость репетиции в этой комнате за 3 часа (integer);
- baseid – id реп. базы, которой принадлежит комната (связь с таблицей rehbase) (FK, один-ко-многим).

Таблица equipment должна содержать информацию о:

- id – идентификатор оборудования (PK);
- type – тип оборудования (text);

- brand – фирма оборудования (text);
- amount – количество оборудования такого типа в соответствующей комнате (integer);
- roomid – id комнаты, в которой находится это оборудование (связь с таблицей room) (FK, один-ко-многим).

Таблица account должна содержать информацию о:

- id – идентификатор пользователя (PK);
- fio – ФИО пользователя (text);
- phone – номер телефона пользователя (text);
- mail – электронная почта пользователя (text);
- password – пароль от аккаунта (text);
- type – тип пользователя (text).

Таблица rehearsal должна содержать информацию о:

- id – идентификатор репетиции (PK);
- rehdate – дата и время репетиции (timestamp);
- musicianid – id музыканта, забронировавшего репетицию (связь с таблицей account) (FK, один-ко-многим);
- roomid – id комнаты, в которой забронирована репетиция (связь с таблицей room) (FK, один-ко-многим).

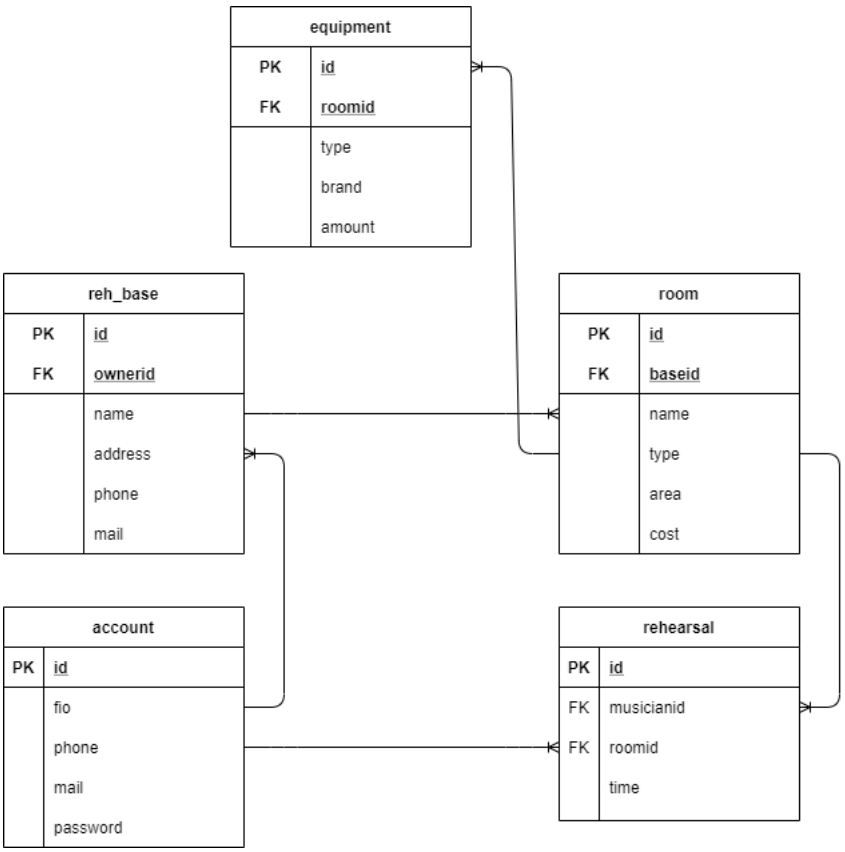

Рисунок 3 – ER диаграмма проектируемой БД

2.2 Требования к программе

Разрабатываемое ПО должно предоставлять следующие возможности:

- регистрация нового пользователя;
- авторизация пользователя;
- вывод списка комнат, доступных для бронирования;
- вывод подробной информации о каждой комнате;
- бронь комнаты на соответствующее время;
- отмена брони;
- вывод уже забронированных (будущих) репетиций;
- вывод зарегистрированных реп. баз для данного аккаунта;
- регистрация новой реп. базы;
- добавление комнаты в зарегистрированную реп. базу;
- добавление оборудования в соответствующую комнату;

- вывод всех будущих репетиций на соответствующей реп. базе;
- удаление реп. базы;
- поиск по имени реп. базы;
- поиск по дате и времени репетиции.

Ограничения, в рамках которых будет работать программа:

- аккаунт нельзя удалить;
- нельзя зарегистрироваться стандартным способом в качестве администратора (администраторы добавляются «вручную» на уровне БД);
- для обновления информации нужно закрыть соответствующее окно и снова его открыть;
- для выхода из аккаунта нужно закрыть окно;
- пароль от аккаунта хранится в БД в обычном виде, без шифрования;
- календарь для бронирования репетиций не предоставляется.

2.3 Структура ПО

Всё проектируемое ПО можно разделить на две основные части:

- frontend (часть приложения, с которой непосредственно взаимодействует пользователь, отображение данных);
- backend (взаимодействие с базой данных).

Структура frontend части в свою очередь будет основана на паттерне MVC (Model, View, Controller).

Таким образом, проект будет разделён на несколько файлов:

- connect – взаимодействие непосредственно с БД;
- gui – здесь будут находиться модели и контроллеры frontend части;
- отдельные файлы для каждого окна итогового приложения, предоставляющие интерфейс пользователю (т. е. отвечающие за view часть).

Файл connect будет состоять из набора следующих функций:

- connect_* – для подключения к БД с соответствующей ролью;
- функции для формирования и посылки запросов к БД.

Файл gui в свою очередь будет состоять из набора классов, каждый из

которых будет соответствовать определённому окну приложения.

На рисунке 4 представлена диаграмма классов разрабатываемого приложения.

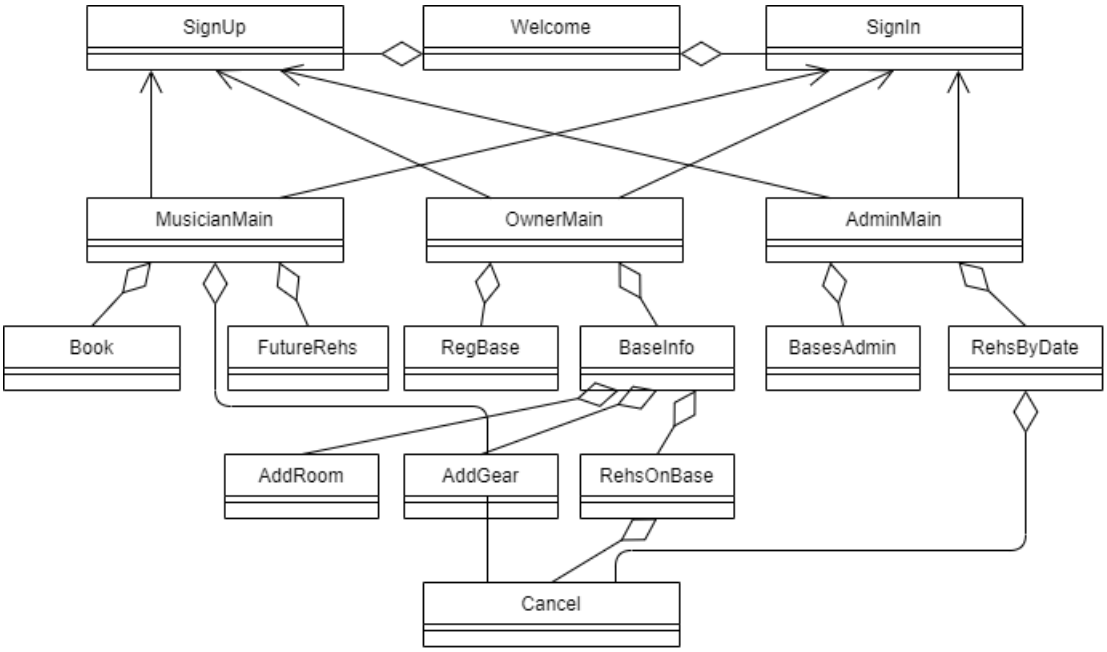

Рисунок 4 – Диаграмма классов

2.4 Способы и этапы тестирования

Для проверки работоспособности ПО будет применяться функциональное тестирование.

Тестирование ПО будет разделено на следующие этапы:

- регистрация нового пользователя (как музыканта, так и владельца);
- попытка зарегистрироваться ещё раз с той же почтой;
- попытка авторизоваться с неправильной почтой или паролем;
- авторизация в качестве музыканта;
  - бронирование репетиции;
  - попытка забронировать репетицию в уже забронированной на это время комнате;
  - просмотр своих будущих репетиций;
  - отмена брони;

  - попытка забронировать репетицию «в прошлом»;
- авторизация в качестве владельца;
  - регистрация новой реп. базы;
  - попытка зарегистрировать уже существующую реп. базу (с той же почтой либо с тем же названием и адресом);
  - добавление комнаты в реп. базу;
  - попытка добавить уже существующую комнату (с тем же названием);
  - добавление оборудования в комнату;
  - попытка добавить оборудование в несуществующую комнату;
  - попытка повторно добавить то же самое оборудование в ту же самую комнату;
  - просмотр предстоящих репетиций на соответствующей реп. базе;
  - удаление реп. базы;
- авторизация в качестве администратора;
  - поиск реп. баз по названию (успешный);
  - попытка найти несуществующую реп. базу;
  - поиск репетиций по дате (успешный);
  - попытка найти несуществующую репетицию.

Выводы

На основе теоретических данных, полученных в аналитическом разделе, были спроектированы база данных и приложение. А также были приведены: требования к программе, способы и этапы тестирования и диаграмма классов.

3 Технологическая часть

3.1 Выбор СУБД

В таблице 3 произведено сравнение наиболее популярных СУБД, которые могут быть использованы для реализации хранения данных в разрабатываемом программном продукте.

Таблица 3 – Особенности различных СУБД

Особенность	**MySQL**	**PostgreSQL**	**MS SQL Server**	**Oracle Database**
Open source	GNU GPL с открытым исходным кодом	Открытый исходный код	Коммерческая	Коммерческая
Соответствие ACID	Частичное (зависит от версии) [4]	Полное	Полное	Полное
NoSQL / JSON	Поддержка некоторых функций	JSON, NoSQL	JSON, NoSQL	JSON, NoSQL
Поддержка MERGE	Да	Нет	Да	Да

Таким образом, для реализации проекта в качестве СУБД будет использоваться PostgreSQL, так как он является свободно распространяемым и удовлетворяет всем необходимым требованиям в рамках поставленной задачи.

3.2 Выбор ЯП и среды программирования

Для реализации проекта в качестве языка программирования был выбран язык Python, так как:

- этот язык поддерживает работу с PostgreSQL (в моём случае для этого будет использоваться библиотека psycopg2);
- данный язык является полностью объектно-ориентированным (всё является объектами) [5], что позволяет в полной мере использовать классы при написании программы;
- в этом языке присутствует PyQt (набор расширений графического фреймворка Qt для языка программирования Python, выполненный в виде расширения Python [6]), позволяющий легко создавать графический интерфейс для приложения.

В качестве среды разработки был выбран PyCharm по следующим причинам:

- наличие бесплатной версии Community Edition;
- множество удобств, облегчающих процесс написания и отладки кода.

3.3 Детали реализации

В приложениях А – Б представлен пример реализации frontend части приложения.

В приложении В представлена реализация соединения с базой данных в соответствии с ролевой моделью.

Помимо этого, на уровне базы данных была реализована процедура, необходимая для удаления реп. базы (приложения Г – Д).

На рисунках ниже представлен основной интерфейс приложения.

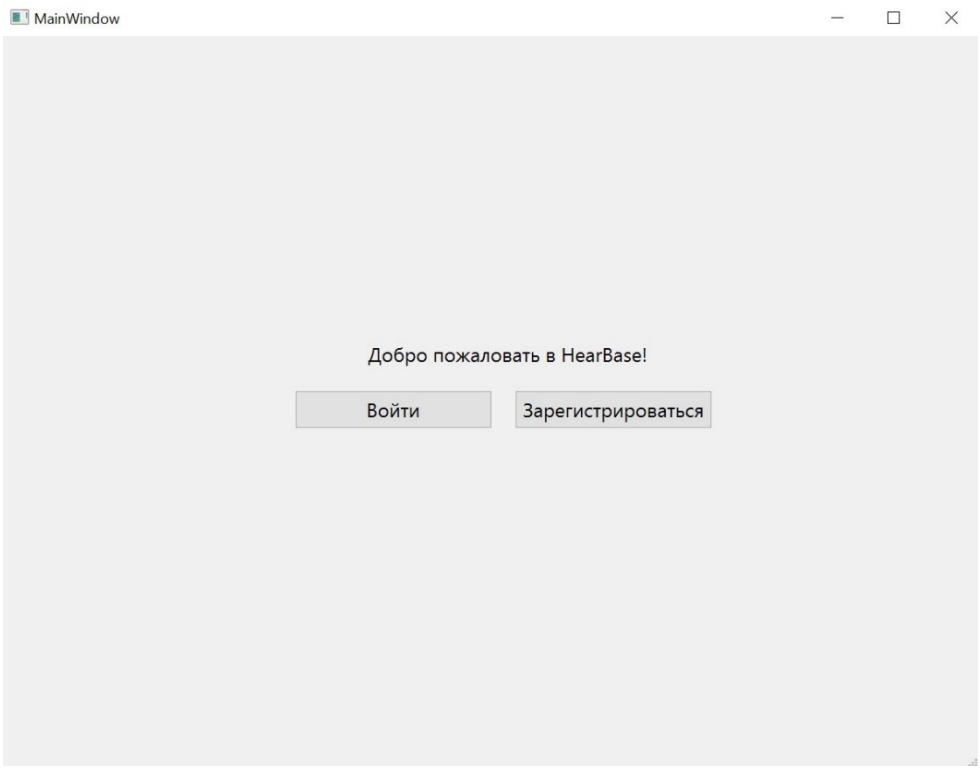

Рисунок 5 – Окно с приглашением

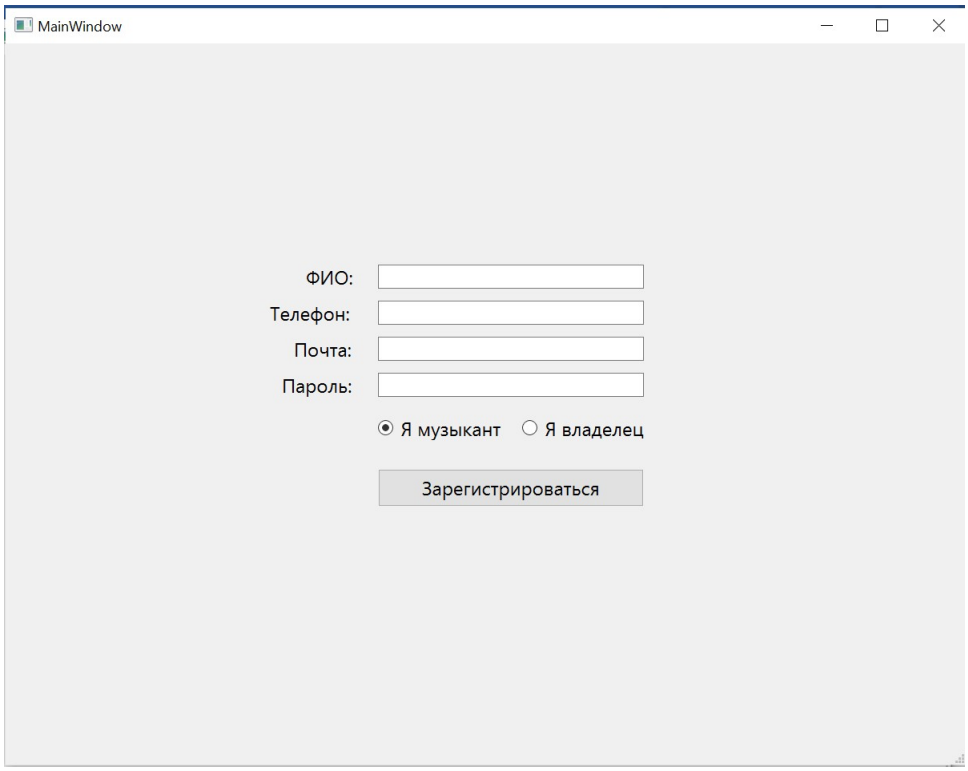

Рисунок 6 – Окно регистрации

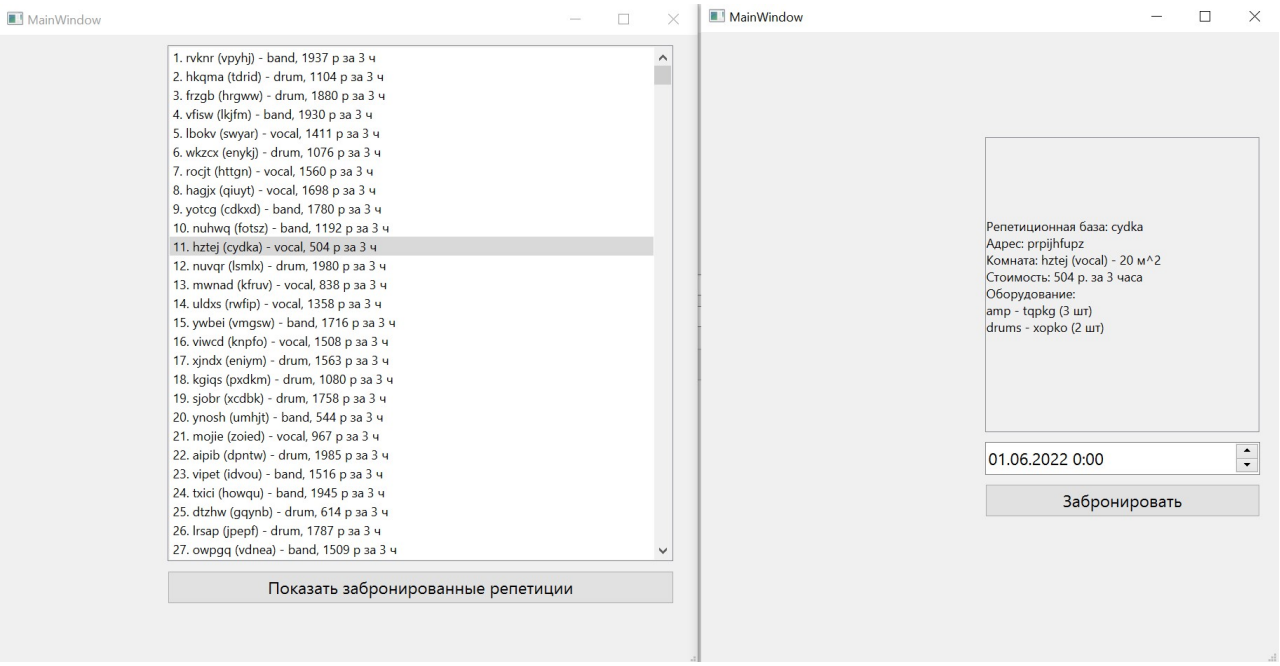

Рисунок 7 – Интерфейс при входе в качестве музыканта

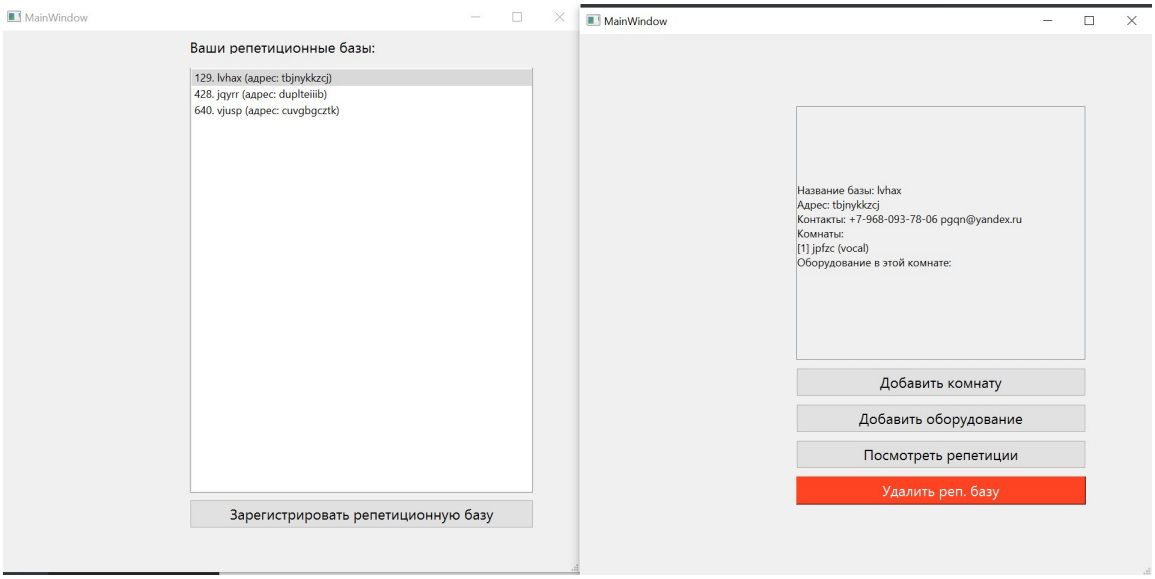

Рисунок 8 – Интерфейс при входе в качестве владельца

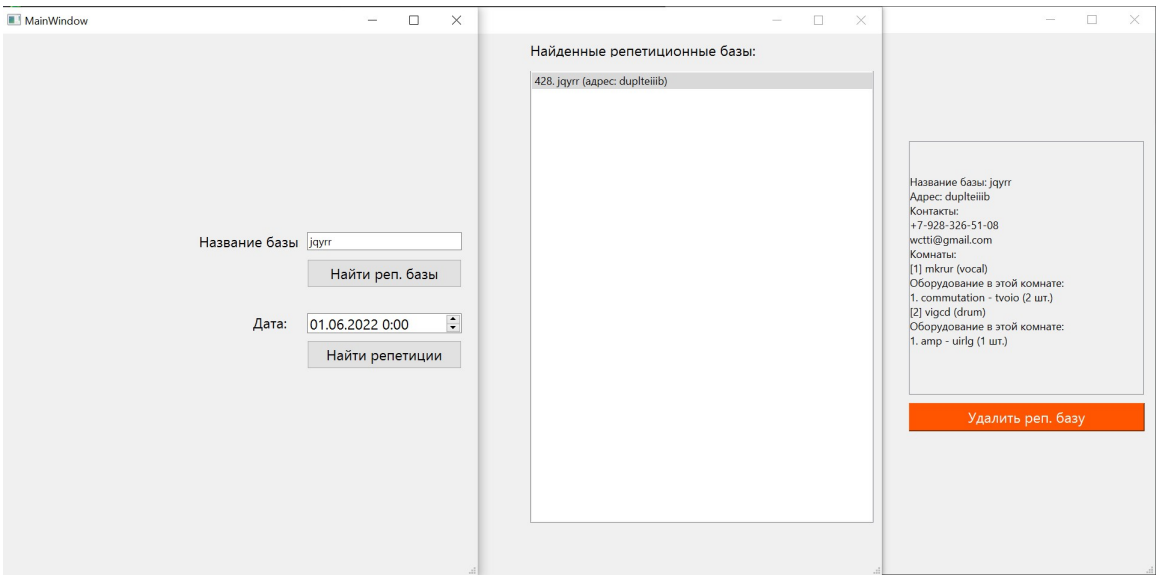

Рисунок 9 – Интерфейс при входе в качестве администратора

Выводы

В данном разделе были выбраны СУБД, язык программирования и среда разработки.

В качестве СУБД выбрана PostgreSQL, так как она:

- свободно распространяемая;
- полностью соответствует требованиям ACID;
- поддерживает все необходимые в рамках поставленной задачи функции (такие, как вложенные селекты, транзакции, триггеры, процедуры).

В качестве языка программирования был выбран Python, так как он:

- поддерживает работу с PostgreSQL;
- объектно-ориентированный;
- обладает необходимыми расширениями для работы с графическим интерфейсом.

В качестве среды разработки был выбран PyCharm, так как он:

- имеет свободно распространяемую версию;
- содержит множество удобств для написания и отладки кода, а также для работы с СУБД.

Помимо этого, в данном разделе был разработан и протестирован исходный код программы. Программа тестировалась в соответствии с этапами, приведёнными в разделе 2.4. Все тесты были успешно пройдены.

4 Исследовательская часть

4.1 Цель эксперимента

Целью эксперимента является выявление зависимости времени выполнения запроса от числа записей в таблице, а также от вида самого запроса.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- выбрать запросы, время выполнения которых будет замеряться;
- измерить время для каждого запроса и каждого числа записей в таблице;
- построить график зависимости времени от размеров таблиц и запросов;
- проанализировать полученные результаты.

Ниже приведены технические характеристики устройства, на котором будет проводиться эксперимент:

- операционная система: Windows 10 64-bit Home;
- оперативная память: 8 GB;
- процессор: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz [**?**].

4.2 Постановка эксперимента

При эксперименте использовались запросы двух видов.

Первый вариант запроса:

```
1  select *
2  from rehearsal join room on rehearsal.roomid = room.id
3  join account on rehearsal.musicianid = account.id
4  join reh_base on room.baseid = reh_base.id
```

Второй вариант запроса:

```
1  select rehearsal.rehdate, room.name, room.type, room.area, room.cost,
2  reh_base.name, reh_base.address, reh_base.phone, reh_base.mail
3  from rehearsal join room on rehearsal.roomid = room.id
4  join account on rehearsal.musicianid = account.id
5  join reh_base on room.baseid = reh_base.id
```

Число записей изменялось последовательно от 1000 до 10000 с шагом

1000. Каждый замер проводился по 100 раз, после чего вычислялось среднее время выполнения. Время измерялось в наносекундах с помощью функции perf_counter_ns() из библиотеки time.

4.3 Результаты эксперимента

По результатам измерений времени выполнения запросов можно составить таблицы 4 – 5 и диаграмму 10.

Таблица 4 – Результаты замеров времени выполнения 1-го запроса

Число записей	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Время	5 350 973	8 883 559	13 577 113	19 818 492	21 778 098	26 708 169	32 112 573	35 606 740	40 054 833	46 621 729

Таблица 5 – Результаты замеров времени выполнения 2-го запроса

Число записей	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
Время	2 801 220	5 349 398	7 868 523	12 793 699	13 863 790	16 516 216	19 407 919	24 735 794	24 548 257	26 687 837

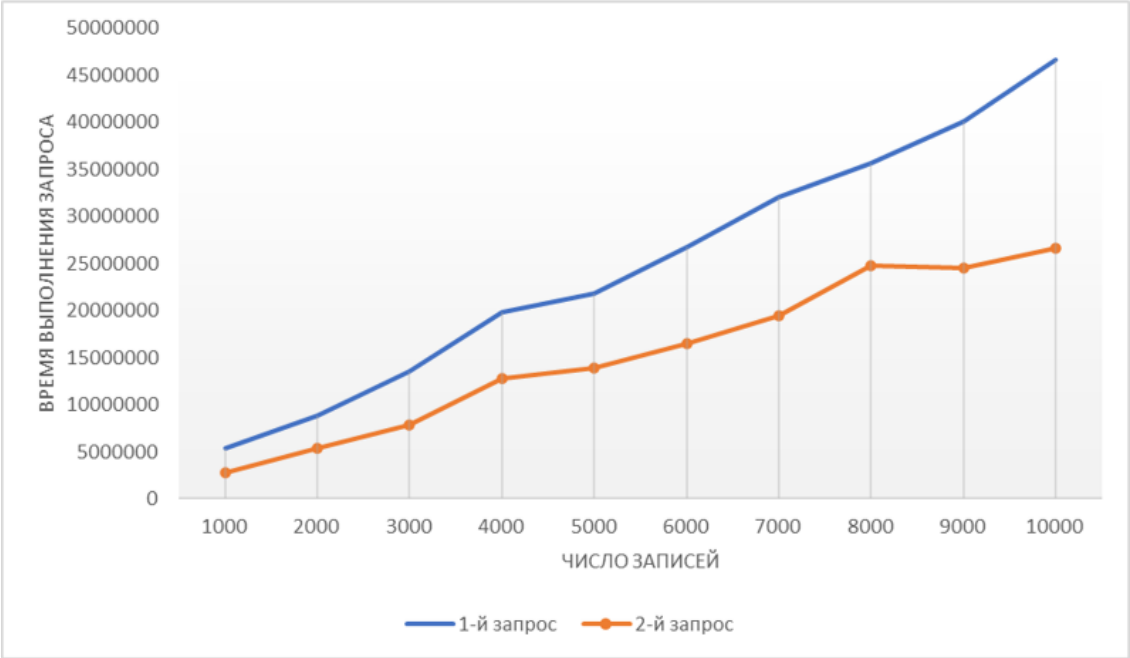

Рисунок 10 – Зависимость времени выполнения запросов от числа записей

Выводы

В данном разделе был проведён анализ времени выполнения двух видов запросов в зависимости от числа записей.

В результате было выяснено, что время выполнения прямо пропорционально числу записей. Также было выяснено, что второй вариант запроса вы-

полняется быстрее первого. За счёт усовершенствования запроса удалось снизить время выполнения в среднем примерно на 40,3%.

Таким образом, на основании полученных данных можно сделать вывод, что на время выполнения запроса влияет как число записей в таблице, так и вид самого запроса.

Заключение

Цель курсового проекта достигнута. Спроектировано и реализовано программное обеспечение для поиска и бронирования репетиционных баз.

В ходе работы было формализовано задание, определён необходимый функционал, проведён анализ различных СУБД, спроектированы и реализованы база данных и приложение в соответствии с поставленной задачей, а также проанализировано время выполнения различных запросов к БД в зависимости от числа записей в таблицах.

Также в ходе выполнения поставленной задачи были изучены возможности языка Python и его расширения PyQt, получен опыт работы с PostgreSQL и pgAdmin, получены знания в области баз данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ISO/IEC TR 10032:2003 Information technology – Reference model of data management.

2. Дейт К. Дж. Введение в системы баз данных. – 8-е изд. – М.: «Вильямс», 2006.

3. Еленев Д. В. и др. Автоматизация системы управления национальным исследовательским университетом и мониторинга его деятельности // Программные продукты и системы, №3, 2012.

4. MySQL и модель ACID [Электронный ресурс]. – Режим доступа: `https://spec-zone.ru/RU/mysql/5.6/storage-engines_mysql-acid.html` (дата обращения: 02.06.2022).

5. Yogesh Rana. Python: Simple though an Important Programming language // International Research Journal of Engineering and Technology (IRJET), 2019.

6. What is PyQt? [Электронный ресурс]. – Режим доступа: `https://riverbankcomputing.com/software/pyqt/intro` (дата обращения: 04.06.2022).

ПРИЛОЖЕНИЕ А

Листинг 1: реализация GUI окна входа/регистрации

```
class Ui_MainWindow(object):
  def setupUi(self, MainWindow):
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(800, 600)
    self.centralwidget = QtWidgets.QWidget(MainWindow)
    self.centralwidget.setObjectName("centralwidget")
    self.signin_button = QtWidgets.QPushButton(self.centralwidget)
    self.signin_button.setGeometry(QtCore.QRect(240, 290, 161, 31))
    font = QtGui.QFont()
    font.setPointSize(12)
    self.signin_button.setFont(font)
    self.signin_button.setObjectName("signin_button")
    self.signup_button = QtWidgets.QPushButton(self.centralwidget)
    self.signup_button.setGeometry(QtCore.QRect(420, 290, 161, 31))
    font = QtGui.QFont()
    font.setPointSize(12)
    self.signup_button.setFont(font)
    self.signup_button.setObjectName("signup_button")
    self.label = QtWidgets.QLabel(self.centralwidget)
    self.label.setGeometry(QtCore.QRect(300, 250, 251, 20))
    font = QtGui.QFont()
    font.setPointSize(12)
    self.label.setFont(font)
    self.label.setObjectName("label")
    MainWindow.setCentralWidget(self.centralwidget)
    self.menubar = QtWidgets.QMenuBar(MainWindow)
    self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 18))
    self.menubar.setObjectName("menubar")
    MainWindow.setMenuBar(self.menubar)
    self.statusbar = QtWidgets.QStatusBar(MainWindow)
    self.statusbar.setObjectName("statusbar")
    MainWindow.setStatusBar(self.statusbar)

    self.retranslateUi(MainWindow)
    QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

ПРИЛОЖЕНИЕ Б

Листинг 2: контроллер окна входа/регистрации

```
class Welcome(QtWidgets.QMainWindow, welcome.Ui_MainWindow):
  def __init__(self):
    super().__init__()
    self.setupUi(self)
    self.signin_button.clicked.connect(self.sign_in)
    self.signup_button.clicked.connect(self.sign_up)

  def sign_in(self):
    self.window = SignIn()
    self.window.show()

  def sign_up(self):
    self.window = SignUp()
    self.window.show()
```

ПРИЛОЖЕНИЕ В

Листинг 3: подключение к БД с разными ролями

```
1     def connect():
2       connection = None
3       try:
4         connection = psycopg2.connect(host="localhost", database="DB_course",
5         user="postgres", password="****")
6       except OperationalError as e:
7         print("The error '{e}' occurred")
8       return connection
9     def connect_musician():
10      connection = None
11      try:
12        connection = psycopg2.connect(host="localhost", database="DB_course",
13        user="musician", password="****")
14        print("Connection of musician successful")
15      except OperationalError as e:
16        print("The error '{e}' occurred")
17      return connection
18    def connect_owner():
19      connection = None
20      try:
21        connection = psycopg2.connect(host="localhost", database="DB_course",
22        user="base_owner", password="****")
23        print("Connection of owner successful")
24      except OperationalError as e:
25        print(f"The error '{e}' occurred")
26      return connection
27    def connect_admin():
28      connection = None
29      try:
30        connection = psycopg2.connect(host="localhost", database="DB_course",
31        user="app_admin", password="****")
32        print("Connection of admin successful")
33      except OperationalError as e:
34        print(f"The error '{e}' occurred")
35      return connection
```

ПРИЛОЖЕНИЕ Г

Листинг 4: процедура удаления реп. базы

```
1     CREATE OR REPLACE PROCEDURE del_all(base_id int)
2     AS $$
3       BEGIN
4       delete from rehearsal where roomid in (select id from room where baseid
      = base_id);
5       delete from equipment where roomid in (select id from room where baseid
      = base_id);
6       delete from room where baseid = base_id;
7       delete from reh_base where id = base_id;
8       END;
9     $$ LANGUAGE PLPGSQL;
```

ПРИЛОЖЕНИЕ Д

Листинг 5: применение процедуры при удалении реп. базы

```
1   def del_base(conn, base_id):
2     cur = conn.cursor()
3     query = "CALL del_all(%s)"
4     cur.execute(query, (base_id,))
5     conn.commit()
6     cur.close()
```