



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

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:

*«Разработка базы данных для хранения треков в
MIDI формате»*

Студент ИУ7-61Б
(Группа)

(Подпись, дата) Е. В. Фролова
(И.О.Фамилия)

Руководитель

(Подпись, дата) А. С. Кострицкий
(И.О.Фамилия)

Оглавление

Введение	3
1 Аналитический раздел	4
1.1 Постановка задачи	4
1.2 Анализ существующих решений	4
1.3 Формализация данных	6
1.4 Типы пользователей	7
1.5 Классификация БД по модели данных	9
1.5.1 Дореляционные БД	10
1.5.2 Реляционные БД	11
1.5.3 Постреляционные БД	12
2 Конструкторский раздел	15
2.1 Проектирование базы данных	15
2.1.1 Таблицы	15
2.1.2 Целостность данных БД	18
2.1.3 Триггеры	19
2.2 Требования к программе	22
3 Технологический раздел	24
3.1 Средства реализации	24
3.1.1 Выбор СУБД	25
3.2 Структура и состав классов	27
3.3 Создание объектов БД	29
3.3.1 Создание таблиц	29
3.3.2 Создание связей	32
3.3.3 Создание ролевой модели	33
3.3.4 Создание триггеров	34
3.4 Интерфейс программы	39

4 Экспериментальный раздел	47
4.1 Сравнение производительности	47
4.1.1 Поиск по первичному ключу	47
4.1.2 Поиск по полю с фильтрацией	48
4.1.3 Объединение таблиц с поиском по внешнему ключу	49
Заключение	52
Список литературы	52

Введение

MIDI (Musical Instrument Digital Interface) — это стандарт обмена данными между цифровыми музыкальными инструментами. Он позволяет обмениваться такой информацией, как номер ноты, скорость нажатия, таймкод и др. Другими словами, MIDI - это формат для хранения нотных партитур в цифровом виде.

Данный формат часто используется во многих генераторах музыкальных произведений, а также среди людей, занимающихся написанием музыки, ее аранжировкой, сведением и звукозаписью. Также MIDI поддерживает большинство выпускаемых музыкальных девайсов.

Цель данной работы - разработать базу данных для хранения треков в формате MIDI и приложение, предоставляющее интерфейс для доступа к БД.

Для достижения заданной цели необходимо выполнить следующие **задачи**:

1. формализовать задание, определить необходимый функционал;
2. провести анализ существующих СУБД и выбрать наиболее подходящую;
3. описать структуру базы данных, включая объекты, из которых она состоит, и связи между ними;
4. создать и заполнить БД;
5. разработать ПО, которое позволит пользователю получать информацию о треках и составлять из них плейлисты;
6. исследовать время и стоимость выполнения запросов к БД с использованием индексов и без.

1 | Аналитический раздел

В данном разделе будет проанализирована поставленная задача и существующие решения, формализованы данные и рассмотрены различные способы хранения данных.

1.1 Постановка задачи

Необходимо разработать базу данных для хранения треков в формате MIDI, а также приложение "библиотеку треков" для доступа к ней. Пользователь должен иметь возможность получать информацию о треках, добавлять их в свою медиатеку и создавать из них плейлисты.

1.2 Анализ существующих решений

Так как формат MIDI является достаточно популярным в области генерации музыкальных произведений и звукозаписи, существует множество сайтов для загрузки бесплатных MIDI-файлов и бесплатных MIDI-песен. Наиболее популярными из них являются:

1. MIDI Stock [1] - огромная и уникальная по содержанию коллекция качественных MIDI файлов и мелодий.
2. BitMidi [2] - 113 241 MIDI-файлов созданных волонтерами со всего мира.
3. Midiworld [3] - Тысячи бесплатных файлов MIDI.

Ниже, на рисунке 1.1, представлена таблица сравнения существующих аналогов.

	MIDI Stock	BitMidi	Midiworld
Описание	Предоставляет для скачивания как оригинальные, так и всевозможные кавер версии треков, а также гитарные, ударные, акустические, бас и соло партии из песен известных исполнителей, групп и музыкальных коллективов.	Предоставляет для скачивания 113 241 MIDI файлов, созданных волонтерами со всего мира.	Предоставляет для скачивания тысячи бесплатных файлов MIDI с треками популярных исполнителей, саундтреками из фильмов/сериалов/игр, а также различные барабанные/пианино паттерны.
Доступность	Бесплатно	Бесплатно	Бесплатно
Возможность отсортировать треки при поиске	-	-	-
Возможность проиграть выбранный MIDI файл	-	+	-
Возможность загрузить свой MIDI файл	+	-	+
Возможность создать плейлист на основе скачанных треков	-	-	-

Рис. 1.1: Сравнение существующих аналогов.

Все вышеперечисленные сайты представляют собой "библиотеку MIDI треков" и позволяют пользователю только скачивать нужные треки, но не предоставляют возможности создавать медиатеку на базе скачанных треков и создавать в ней различные плейлисты.

Таким образом, отличием данной работы от существующих решений является создание плейлистов на базе скачанных MIDI треков.

1.3 Формализация данных

База данных должна хранить информацию о:

- треке;
- пользователях;
- плейлистах.

Таблица 1.1: Категории и сведения о данных

Категория	Сведения
Трек	Название, длительность, дата добавления, описание, Midi файл.
Плейлист	Название, владелец плейлиста, дата создания, описание, избранные треки.
Пользователь	Имя, Страна, Логин, пароль, почта, дата рождения, права доступа

Ниже, на рисунке 1.2, приведена ER-диаграмма сущностей разрабатываемой системы в нотации Чена.

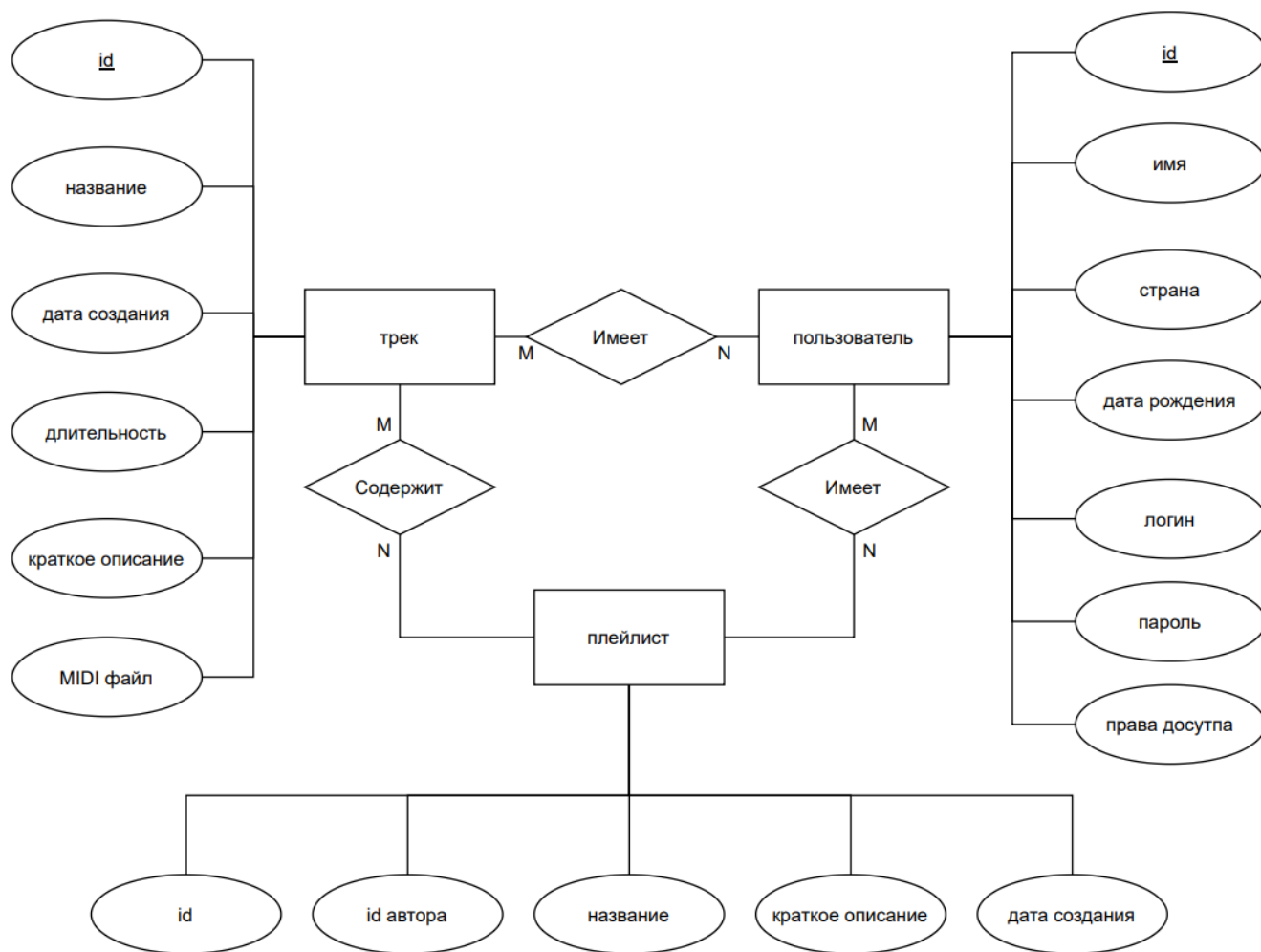


Рис. 1.2: ER-диаграмма сущностей.

1.4 Типы пользователей

Для создания личного списка треков необходима авторизация пользователей. Это делит пользователей на авторизованных и неавторизованных. Для управления библиотекой MIDI треков введены также роли модератора и администратора.

Таблица 1.2: Типы пользователей и доступный им функционал

Тип пользователя	Функционал
Неавторизированный	Регистрация, авторизация.
Авторизованный	Просмотр информации о треках, добавление их в свою медиатеку, создание плейлистов из добавленных в медиатеку треков.
Модератор	Просмотр информации о треках, добавление их в свою медиатеку, создание плейлистов из добавленных в медиатеку треков. Добавление новых треков в "библиотеку Midi треков" и изменение различной информации в ней.
Администратор	Просмотр информации о треках, добавление их в свою медиатеку, создание плейлистов из добавленных в медиатеку треков. Добавление новых треков в "библиотеку Midi треков" и изменение различной информации в ней. Изменение прав доступа пользователей.

Ниже, на рисунках 1.3 - 1.4, представлена Use-Case диаграмма.

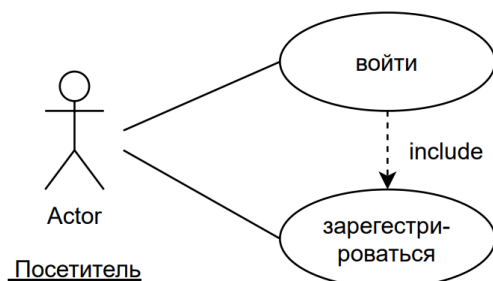


Рис. 1.3: Use-Case диаграмма

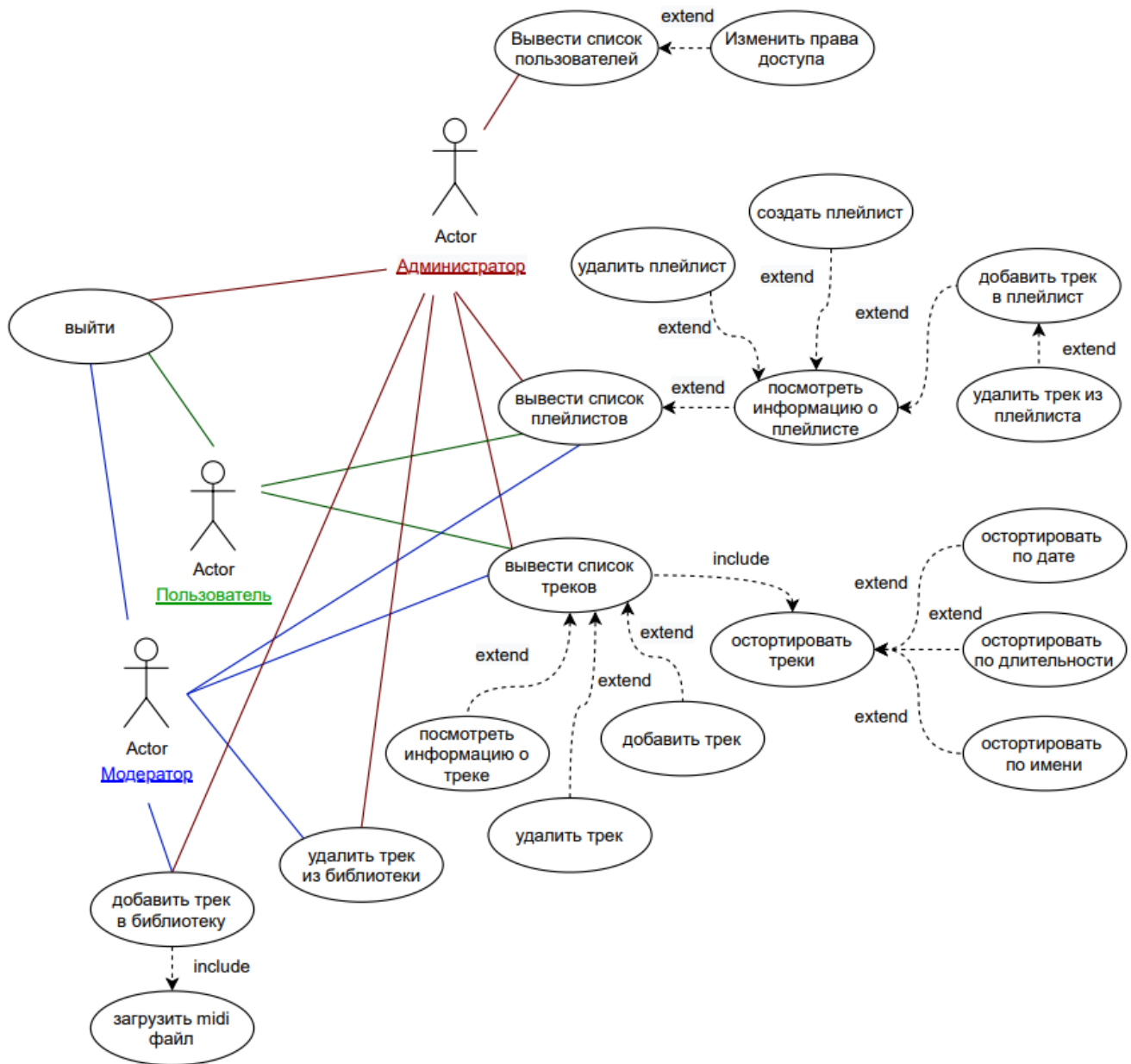


Рис. 1.4: Use-Case диаграмма

1.5 Классификация БД по модели данных

Существует 3 основных типа моделей организации данных:

1. дореляционная;
 - (а) иерархическая;
 - (b) сетевая.
2. реляционная;

3. постреляционная.

1.5.1 Дореляционные БД

В **иерархической** модели данных используется представление базы данных в виде древовидной структуры, состоящей из объектов различных уровней [4]. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка к потомку, при этом возможна ситуация, когда объект-предок имеет несколько потомков, тогда как у объекта-потомка обязателен только один предок.

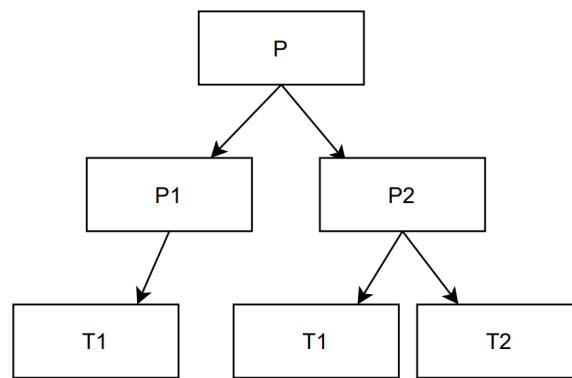


Рис. 1.5: структура иерархической модели данных

Для БД определяется полный порядок обхода - сверху вниз и слева направо.

Сетевая модель данных представляет собой расширение иерархической. В иерархических моделях потомок может иметь только одного предка, а в сетевой потомок может иметь любое количество предков. Сетевая БД состоит из набора экземпляров определенного типа записи и набора экземпляров определенного типа связей между этими записями. Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка.

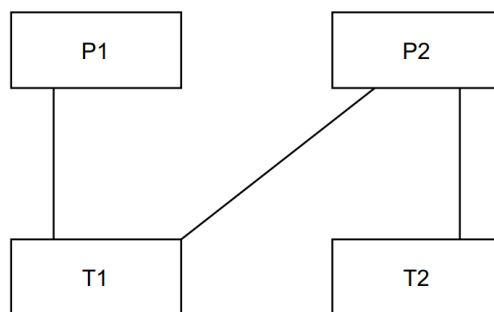


Рис. 1.6: структура сетевой модели данных

К **преимуществам** дореляционных моделей данных относят экономию памяти и оперативность выполнения операций над данными.

Недостатками таких моделей являются сложность использования, зависимость прикладных систем от физической организации (т.к. логика процедуры выбора данных зависит от физической организации этих данных, то данная модель не является полностью независимой от приложения) и как следствие перегруженность логики деталями организации доступа к БД.

1.5.2 Реляционные БД

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

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. В каждой таблице должно быть хотя бы одно ключевое поле, содержимое которого уникально для любой записи в этой таблице. Значения ключевого поля однозначно определяют каждую запись в таблице. За счет возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов.

При использовании реляционной модели данных предполагается неделимость данных, хранящихся в полях записей таблиц. Также одним из основных положений реляционной модели данных является требование нормализации отношений.

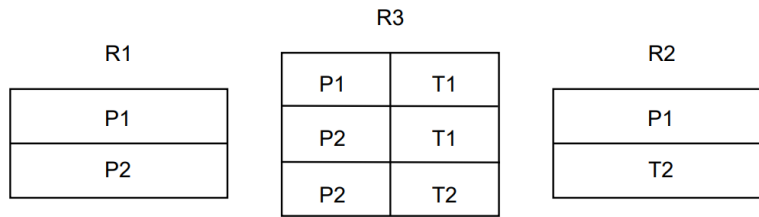


Рис. 1.7: структура реляционной модели данных

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

Преимуществами такой модели являются простота и наглядность (единственной используемой информационной конструкцией является таблица), полная независимость данных (изменения в прикладной программе при изменении БД минимальны), для разработки ПО нет необходимости знать конкретную организацию БД во внешней памяти.

Недостатками реляционной модели являются [5]:

1. Строгая типизация, приводящая к несоответствию структуры БД структуре данных реального объекта. Для хранения в реляционной базе данные одного информационного объекта должны быть декомпозированы и распределены по множеству равноценных нормализованных таблиц.
2. Отдельное от информационного объекта хранение и выполнение его собственных действий. Поведение объекта в РБД описывается в виде хранимых в базе функций, процедур и триггеров, не принадлежащих информационному объекту.
3. Плохая масштабируемость, вызывающая стремительное падение производительности при росте объема данных и количества используемых в запросах соединений (JOIN) таблиц.
4. Неустойчивость к отказам оборудования.

1.5.3 Постреляционные БД

Появление постреляционных БД обусловлено необходимостью исправить недостатки реляционной модели данных, которые были перечислены в выше, в разделе 1.5.2.

Ниже, на рисунке 1.8, представлена классификация постреляционных баз данных.

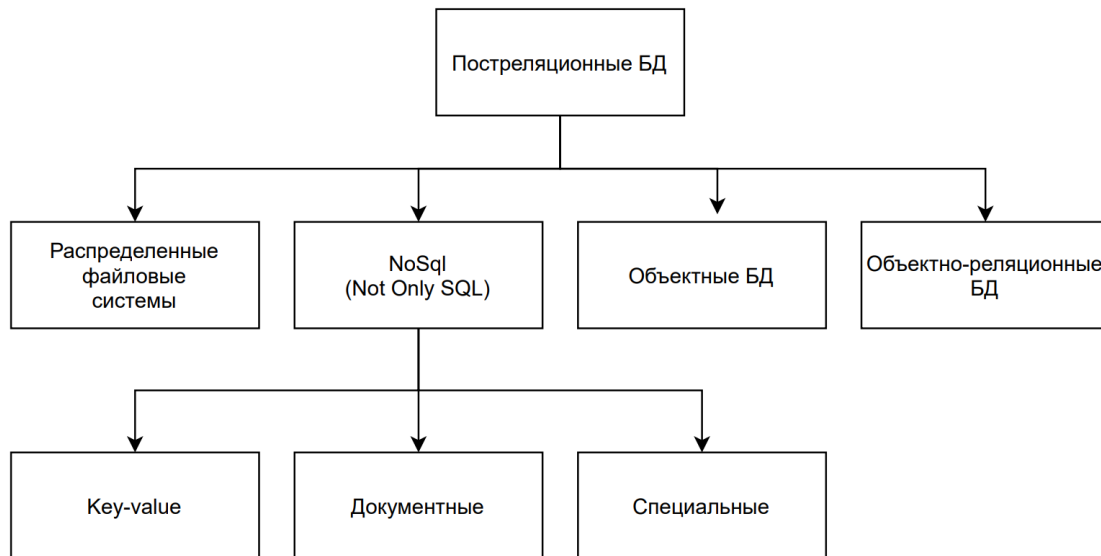


Рис. 1.8: Классификация постреляционных БД

Распределенные ФС

Необходимость совместного выполнения противоречивых требований масштабируемости, эффективности и отказоустойчивости обусловила создание и использование средств для распределенного хранения и обработки данных параллельно во многих узлах вычислительной сети. Распределенные ФС решают задачу хранения растущего объема данных, они позволяют преодолеть ограниченность объема памяти, вычислительной мощности и надежности одного компьютера.

Объектные БД

Появление объектно-ориентированных БД (ООБД) обусловлено стремлением сблизить представления данных в базе и обрабатывающей программе на объектно-ориентированном языке. ООБД обеспечивают создание стабильных — долговременно хранящихся — программных объектов и средства для поиска и управления этими объектами.

Объектно-реляционные БД

Требование нормализации отношений в реляционной модели не позволяет представлять в рамках одной таблицы информационные объекты, имеющие

сложную внутреннюю структуру. Объектно-реляционная модель расширяет реляционный подход системой новых структурных типов данных. Среди новых типов появились как специальные (геометрические, географические), так и универсальные типы (массивы, коллекции, объекты).

NoSQL БД

NoSQL БД используют новые, как табличные, так и не табличные, структуры данных, язык запросов которых не является SQL. Отказ от SQL продиктован необходимостью повысить эффективность запросов за счет исключения длинных цепочек соединений таблиц или переходом к другим (не табличным) структурам данных.

Постреляционные БД также можно классифицировать по способу размещения данных в устройствах хранения:

- хранилища In memory: информационные объекты постоянно хранятся в оперативной памяти серверов, образующих кластер;
- централизованные или распределенные БД — находящиеся в одном устройстве или на множестве взаимодействующих устройств;
- облачные хранилища данных, размещаемые в датацентрах IT-компаний.

Вывод

В данном разделе была проанализирована поставленная задача и рассмотрены способы ее реализации. На основе анализа существующих моделей баз данных было решено использовать в данной работе реляционную СУБД.

В данной работе существует необходимость хранить в БД MIDI файл, связанный с треком. Сам файл будет храниться в файловой системе на сервере, а ссылка на него (его расположение) — в БД в качестве текстового поля.

2 | Конструкторский раздел

В данном разделе будет спроектированы база данных и приложение для доступа к ней.

2.1 Проектирование базы данных

2.1.1 Таблицы

База данных должна хранить рассмотренные в таблице 1.1 данные. В БД будут существовать 3 сущности и 5 таблиц:

- Таблица треков Tracks;
- Таблица пользователей Users;
- Таблица плейлистов Playlists;
- Таблица TP для формализации связи между Треком и Плейлистом;
- Таблица TU для формализации связи между Треком и Пользователем.

Ниже, на рисунке 2.1, представлена диаграмма разрабатываемой базы данных.

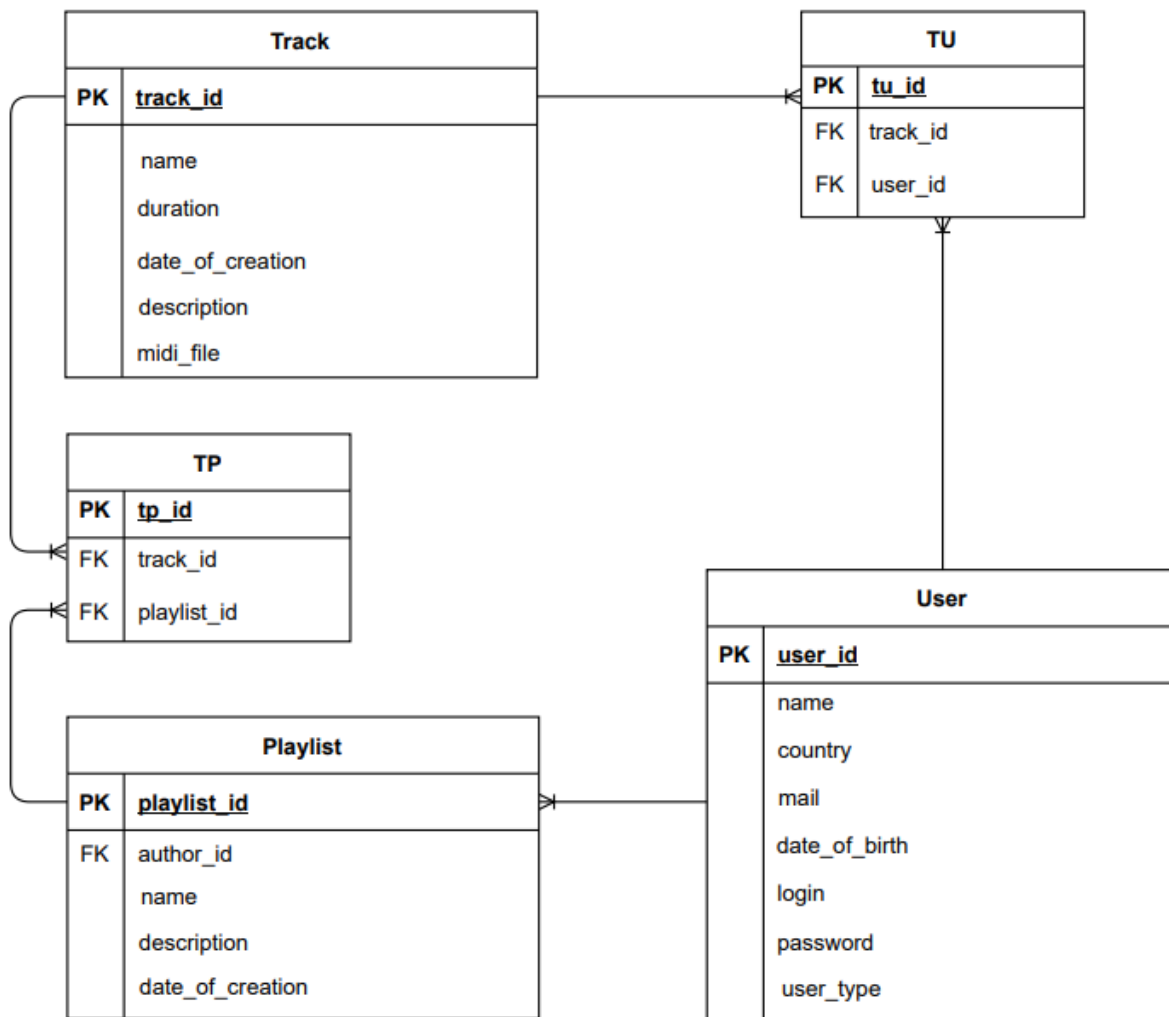


Рис. 2.1: Диаграмма БД.

Поля таблицы **Users** означают:

- user_id - уникальный идентификатор пользователя, первичный ключ, INT PK;
- name - имя пользователя, VARCHAR(50);
- country - страна, VARCHAR(50);
- mail - почта пользователя, VARCHAR(50);
- date_of_birth - дата рождения пользователя, DATE;
- login – логин пользователя, используется для авторизации, VARCHAR(50) UNIQUE;

- password - пароль пользователя, используется для авторизации, VARCHAR(50);
- user_type - тип прав доступа пользователя (0 - администратор, 1 - модератор, 2 - обычный пользователь), INT;

Таблица **Tracks** хранит информацию о треках:

- track_id - уникальный идентификатор трека, первичный ключ, INT PK;
- name - название трека, VARCHAR(50);
- duration - продолжительность трека, TIME;
- date_of_creation - дата создания трека, DATE;
- description - краткое описание трека, TEXT;
- midi_file - расположение файла трека на сервере, TEXT.

Таблица **Playlists** хранит информацию о плейлистах:

- playlist_id - уникальный идентификатор плейлиста, первичный ключ, INT PK;
- author_id - идентификатор пользователя, который создал этот плейлист, внешний ключ (таблица Users, поле user_id), INT FK;
- name - название плейлиста, VARCHAR(50);
- description - краткое описание плейлиста, TEXT;
- date_of_creation - дата создания плейлиста, DATE.

Таблицы TP (Track - Playlist) и TU (Track - User) нужны для формализации связей "многие ко многим".

Таблица **TP** связывает плейлист и треки, входящие в данный плейлист:

- tp_id - уникальный идентификатор связи, первичный ключ, INT PK;
- track_id - идентификатор трека, внешний ключ, INT FK;
- playlist_id - идентификатор плейлиста, внешний ключ, INT FK.

Таблица **TU** связывает пользователя и треки, добавленные в медиатеку пользователя:

- `tu_id` - уникальный идентификатор связи, первичный ключ, INT PK;
- `track_id` - идентификатор трека, внешний ключ, INT FK;
- `user_id` - идентификатор пользователя, внешний ключ, INT FK.

2.1.2 Целостность данных БД

Целостность таблиц

Для обеспечения целостности таблиц необходимо, чтобы все строки в таблице имели уникальный идентификатор (первичный ключ).

Как было описано выше, в разделе 2.1.1, каждая таблица имеет первичный ключ:

1. у таблицы `Users` - поле `user_id`;
2. у таблицы `Tracks` - поле `track_id`;
3. у таблицы `Playlists` - поле `playlist_id`;
4. у таблицы `Tr` - поле `tp_id`;
5. у таблицы `TU` - поле `tu_id`.

Целостность ссылок

Для обеспечения ссылочной целостности необходимо создать внешние ключи, которые поддерживает согласованное состояние данных между двумя таблицами (таблицы, на которую ссылаются, и таблицы, которая ссылается на другую).

1. Между таблицами `Users` и `Playlists` существует связь "один ко многим", для ее формализации создается один внешний ключ: поле `author_id` таблицы `Playlists` ссылается на первичный ключ таблицы `Users` (поле `id`).
2. Таблица `TP` нужна для формализации связи "многие ко многим" между таблицами `Tracks` и `Playlists`, поэтому в ней создаются два внешних ключа:
 - поле `track_id` ссылается на первичный ключ таблицы `Tracks` (поле `id`);

- поле `playlist_id` ссылается на первичный ключ таблицы `Playlists` (поле `id`).

3. Таблица `TU` нужна для формализации связи "многие ко многим" между таблицами `Tracks` и `Users`, поэтому в ней создаются два внешних ключа:

- поле `track_id` ссылается на первичный ключ таблицы `Tracks` (поле `id`);
- поле `user_id` ссылается на первичный ключ таблицы `Users` (поле `id`).

Также для обеспечения ссылочной целостности необходимо, чтобы строка основной таблицы, на которую ссылаются, не могла быть удалена, если есть внешние ключи, ссылающиеся на эту строку, т.е. пока не будет уничтожена связь. Поэтому для таблиц `Users`, `Tracks` и `Playlists` необходимо создать триггеры на удаление данных: при удалении записи в основной таблице удаляются все записи с указанным ключом из зависимых таблиц. Описание соответствующих триггеров представлено ниже, в разделе 2.1.3.

Целостность полей

Для обеспечения целостности полей необходимо указать набор значений данных, которые являются допустимыми для поля, и определить, возможно ли использование нулевого значения.

1. Таблица **Users**. Для полей `name`, `mail`, `date_of_birth`, `login`, `password` и `user_type` не допустимо нулевое значение; поле `country` может быть нулевым. Поле `login` также должно быть уникальным (т.к. авторизация пользователя осуществляется по его логину). Поле `user_type` может принимать только значения 0, 1, 2.
2. Таблиц **Tracks**. Для полей `name`, `duration`, `date_of_creation`, `midi_file` не допустимо нулевое значение; поле `description` может быть нулевым.
3. Таблиц **Playlists**. Для полей `name`, `date_of_creation` не допустимо нулевое значение; поле `description` может быть нулевым.

2.1.3 Триггеры

Триггеры на вставку и обновление

Для таблицы `Users` необходимо реализовать триггеры на вставку и обновление данных:

- При добавлении нового пользователя необходимо наделить его правами доступа;
- При обновлении роли пользователя (поле `user_type`) необходимо обновить его права доступа к таблицам БД.

Ниже, на рисунке 2.2, представлены схемы вышеперечисленных триггеров.

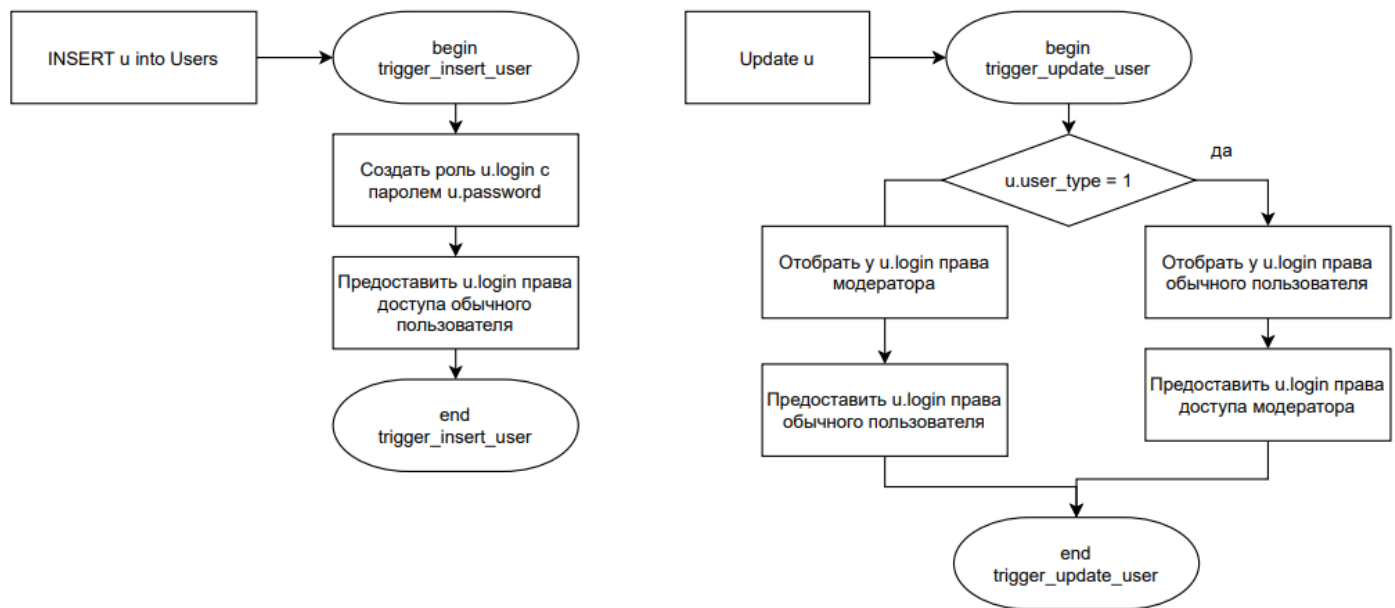


Рис. 2.2: Схемы триггеров.

Триггеры на удаление

Как было сказано выше, в разделе 2.1.2, для обеспечения целостность ссылок для таблиц `Users`, `Tracks` и `Playlists` необходимо создать триггеры на удаление данных.

1. Для таблицы `Users` - при удалении пользователя удаляются все плейлисты и треки из его медиатеки, т.е. при удалении строки из таблицы `Users` удаляются все строки из таблиц `Playlists`, `Tu`, `Tr`, зависящие от удаляемой строки.
2. Для таблицы `Tracks` - при удалении трека из "библиотеки треков он удаляется из медиатек всех пользователей, т.е. при удалении строки из таблицы `Tracks`, удаляются все строки из таблиц `TU` и `TP`, которые зависят от удаляемой строки.

3. Для таблицы Playlists - при удалении плейлиста удаляются все записи из таблицы TP, связанные с этим плейлистом, т.е. при удалении строки из таблицы Playlists, удаляются все строки из таблицы TP, зависящие от удаляемой строки.

Ниже, на рисунках 2.3 - 2.4, представлены схемы вышеперечисленных триггеров.

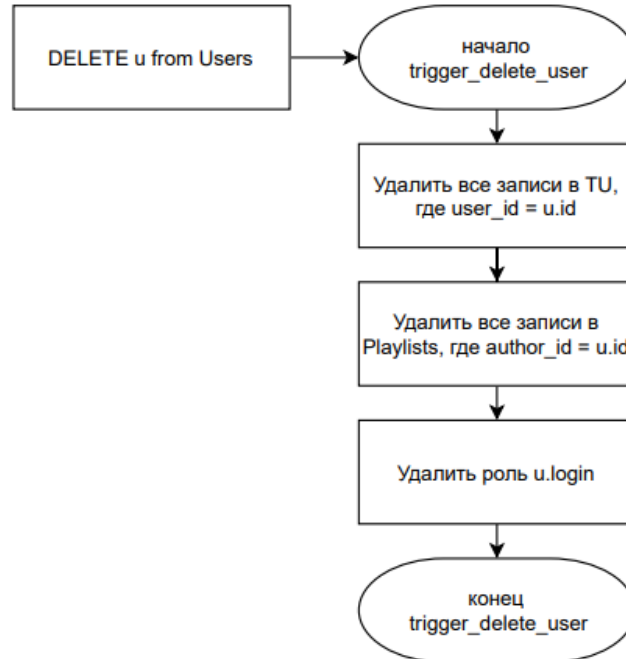


Рис. 2.3: Схемы триггеров.

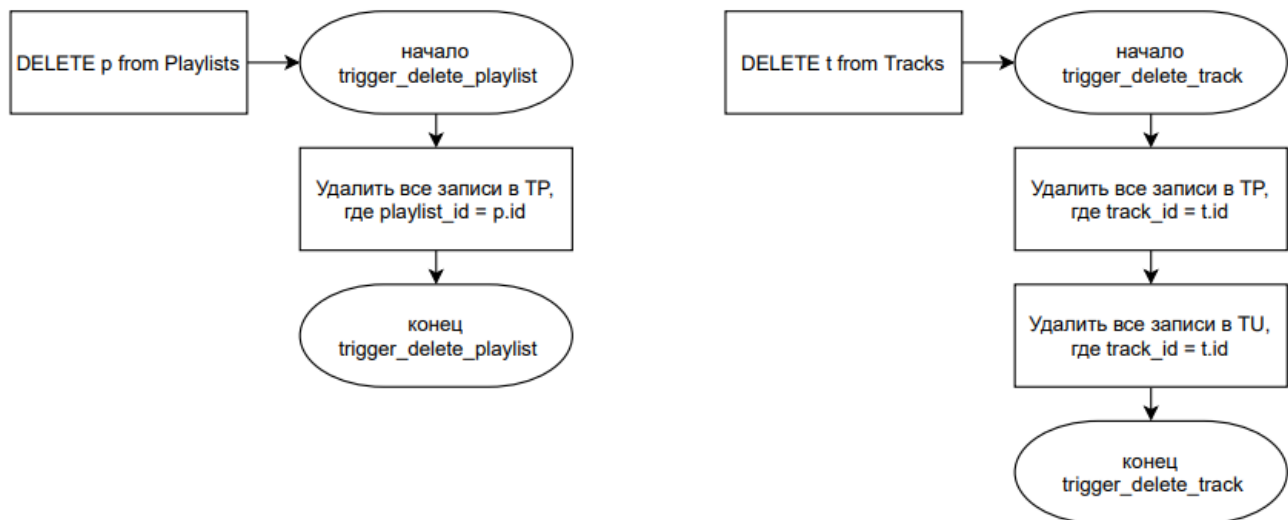


Рис. 2.4: Схемы триггеров.

2.2 Требования к программе

Для доступа к спроектированной базе данных необходимо разработать приложение, предоставляющее интерфейс к БД. В соответствии с Use-Case диаграммой, представленной ранее на рисунках 1.2 - 1.3, программа должна предоставлять следующие возможности.

Возможности обычного пользователя:

- регистрация;
- авторизация;
- вывод всех треков пользовательской медиатеки;
- добавление трека в медиатеку;
- удаление трека из медиатеки;
- вывод информации о конкретном треке;
- вывод всех плейлистов пользовательской медиатеки;
- создание плейлиста в медиатеке;
- удаление плейлиста из медиатеки;
- добавление треков в плейлист;
- вывод информации о конкретном плейлисте;
- вывод информации о профиле пользователя.

Для **модератора** добавляются следующие возможности:

- Добавление трека в библиотеку MIDI треков;
- Удаление трека из библиотеки MIDI треков;
- Изменение названия/описания треков в библиотеке.

Администратор должен иметь возможность изменять права доступа пользователей:

- Сделать обычного пользователя модератором;
- Отобрать права у модератора и сделать его обычным пользователем.

Вывод

В данном разделе были спроектированы база данных и приложение для доступа к ней.

3 | Технологический раздел

В данном разделе представлены архитектура приложения, средства разработки программного обеспечения, детали реализации и способы взаимодействия с программным продуктом.

3.1 Средства реализации

В качестве **языка программирования** был выбран C# [6], так как:

- он является объектно-ориентированным, что позволит:
 - использовать наследование, абстрактные классы и т.д.;
 - использование классов позволит легко организовать взаимодействие между различными объектами, положительно влияя на читабельность кода.
- имеется навыки использования данного языка программирования.

В качестве **среды разработки** была выбрана «Visual Studio 2022» [7] по следующим причинам:

- она распространяется бесплатно для студентов;
- она имеет множество удобств, которые облегчают процесс написания и отладки кода;
- она обеспечивает работу с Windows Forms – интерфейсом, который упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обертки для существующего Win32 API в управляемом коде;
- имеются навыки использования данной среды разработки.

3.1.1 Выбор СУБД

Ниже приведен аналитический обзор популярных реляционных СУБД [8], по результатам которого будет выбрана наиболее подходящая для данной работы СУБД.

Доступность

1. Oracle - коммерческая лицензия.
2. MySQL - имеет двойное лицензирование.
3. Microsoft SQL Server - пользовательское соглашение Microsoft EULA.
4. PostgreSQL - свободная.

Функциональность

1. Oracle:
 - многоверсионность данных для управления параллельными транзакциями;
 - пакеты;
 - аналитические функции в SQL;
 - объектные типы;
 - автоматический мониторинг и диагностика баз для выявления проблем производительности и возможность автоматической корректировки.
2. MySQL:
 - поддержка полусинхронного механизма репликации;
 - дополнительный набор функций для обработки XML;
 - API для плагинов, позволяющий загружать сторонние модули без перезапуска сервера;
 - соединения клиент-сервер, защищенные через SSL.
3. Microsoft SQL Server:
 - встроенный механизм репликации;
 - поддержка JSON, XML;

- поддержка всех существующих драйверов и фреймворков;
- улучшенная производительность за счёт заранее скомпилированных модулей Transact-SQL.

4. PostgreSQL:

- многоверсионность данных для управления параллельными транзакциями;
- возможность написания функций на таких языках, как C, C++, Java, а также ряде скриптовых языков;
- расширенный набор встроенных типов данных;
- возможность создания пользовательских типов и программировать для них механизмы индексирования.

Вывод

Ниже, на рисунке 3.1, представлена таблица сравнения реляционных СУБД по необходимым в данной работе параметрам.

	Oracle	MySQL	MS SQL Server	PostgreSQL
Тип БД	Мульти модельная, SQL	SQL	T-SQL	Объектно- реляционная, SQL
Доступность	Коммерческая лицензия	Имеет двойное лицензирование	Пользовательское соглашение Microsoft EULA	Свободная
Масштабируемость	Вертикальная	Вертикальная, сложная	Вертикальная, сложная	Вертикальная
Документация	+++	++	+++	++
Пользовательские функции	+	+	+	+
Оконные функции	+	-	+	+
Триггеры	+	+	+	+
Индексы	B-Tree index, Bitmap index, Reverse index, Inverted index, Function based index	B-Tree index, Inverted index	B-Tree index, Hash index, Bitmap index, Inverted index, Partial index, Function based index	B-Tree index, Inverted index, Partial index, Function based index

Рис. 3.1: Сравнение РСУБД.

В данной работе в качестве **СУБД** будет использоваться PostgreSQL [6], так как он:

- распространяется свободно;
- поддерживает сложные структуры и широкий спектр встроенных и определяемых пользователем типов данных;
- предоставляет обширный функционал, включая создание пользовательских функций, процедур, триггеров и возможности индексирования.

3.2 Структура и состав классов

Ниже, на рисунках 3.2 - 3.3, представлена UML-диаграмма классов разработанного приложения.

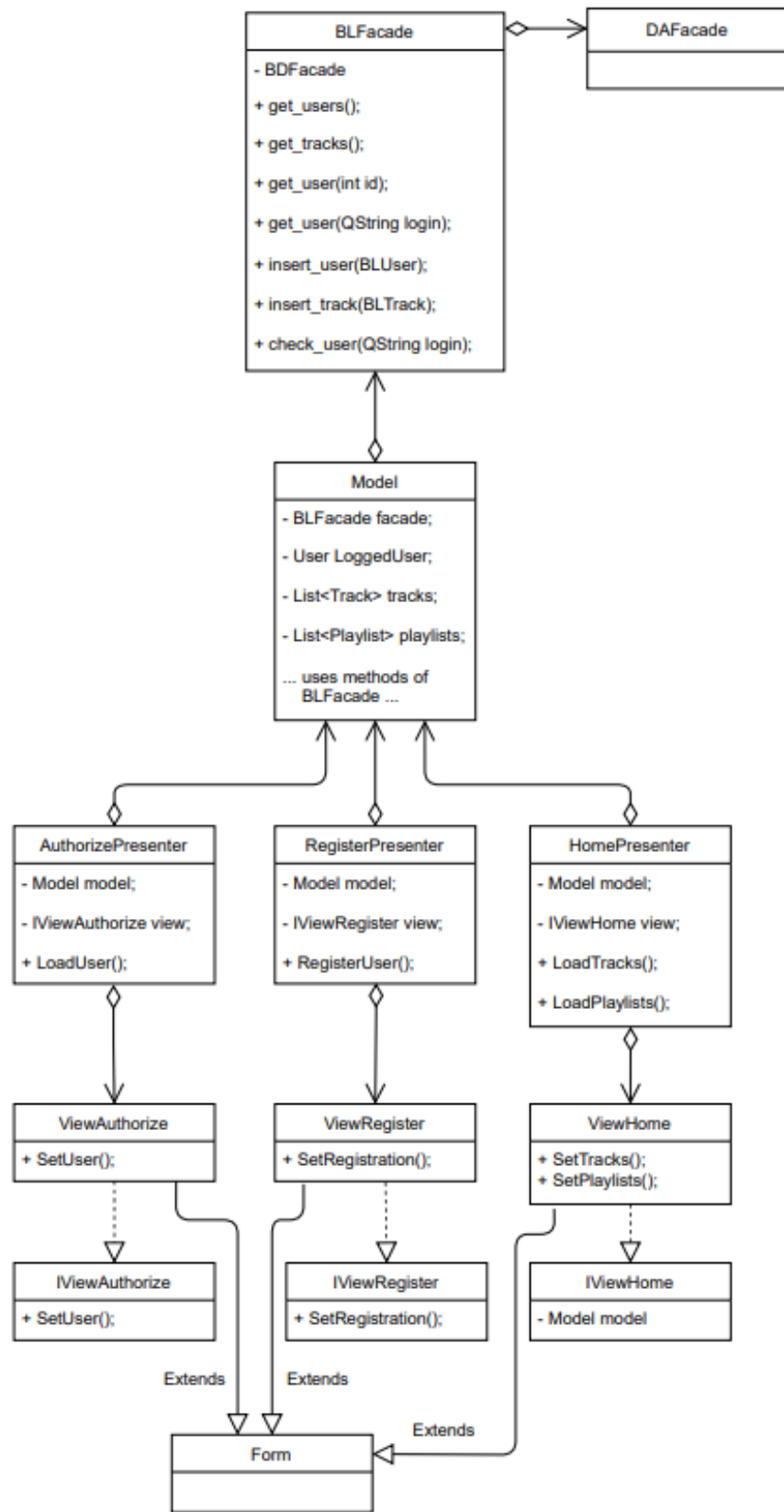


Рис. 3.2: UML-диаграмма

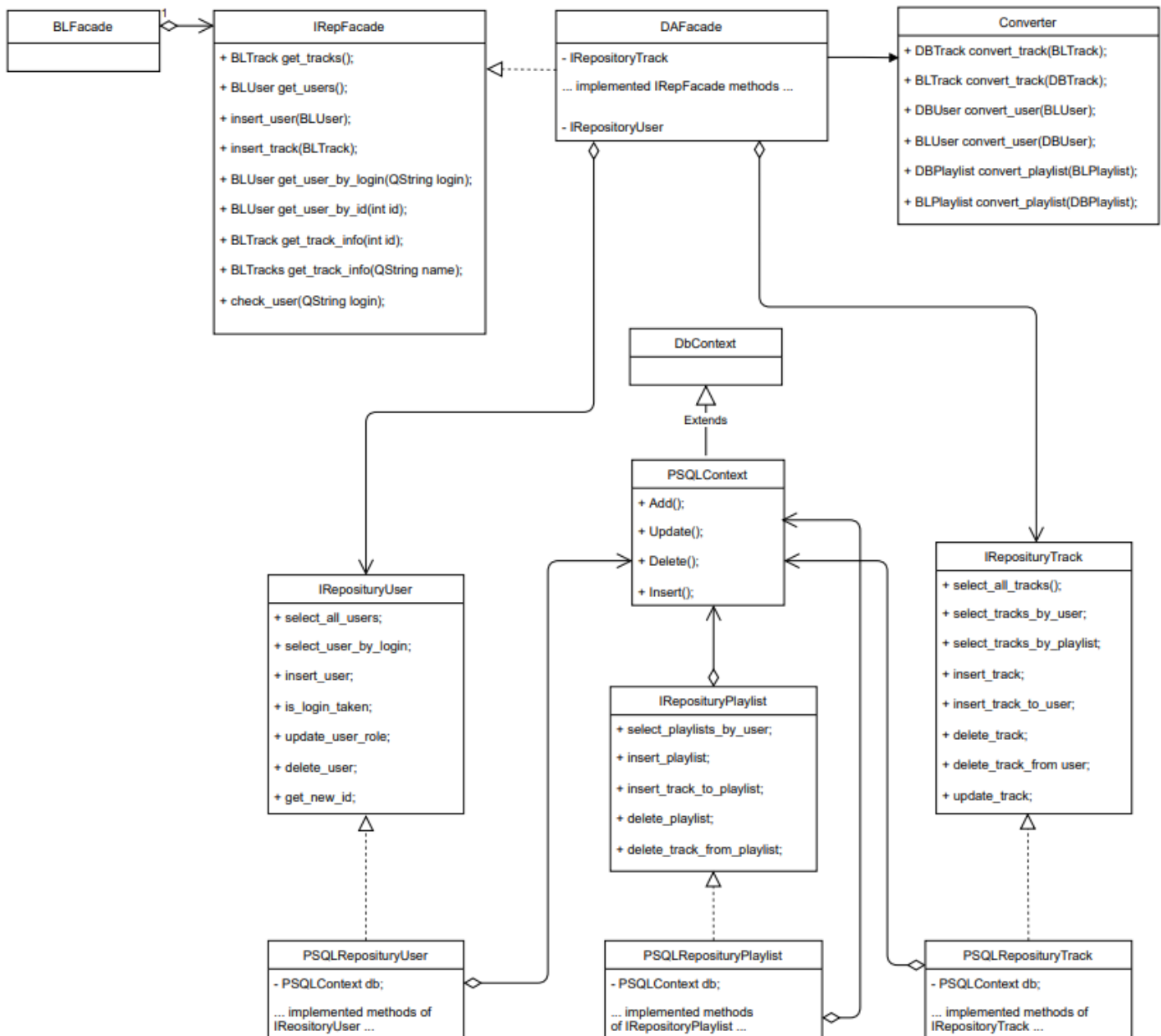


Рис. 3.3: UML-диаграмма

3.3 Создание объектов БД

3.3.1 Создание таблиц

Ниже, на листинге 3.1, представлено создание таблицы Users. Поле id является уникальным идентификатором пользователя, поэтому на него накладывается ограничение Primary Key. На поля name, country, mail, date_of_birth, login, password, user_type накладываются ограничения not null. Также на поле

login накладывается ограничение unique, а на поле user_type ограничение check.

Листинг 3.1: Создание таблицы Users

```
1 create table if not exists Users
2 (
3     id int primary key,
4     name varchar(50) not null,
5     country varchar(50) not null,
6     mail varchar(50) not null,
7     date_of_birth date,
8     login varchar(50) not null unique,
9     password varchar(50) not null,
10    user_type int not null,
11    check (user_type = 0 or user_type = 1
12           or user_type = 2)
13 );
```

Ниже, на листинге 3.2, представлено создание таблицы Tracks. Поле id является уникальным идентификатором трека, поэтому на него накладывается ограничение Primary Key. На поля name, duration, date_of_creation, midi_file накладываются ограничения not null.

Листинг 3.2: Создание таблицы Tracks

```
1 create table if not exists Tracks
2 (
3     id int primary key,
4     name varchar(50) not null,
5     duration time not null,
6     date_of_creation date not null,
7     description text,
8     midi_file text not null
9 );
```

Ниже, на листинге 3.3, представлено создание таблицы Playlists. Поле id является уникальным идентификатором плейлиста, поэтому на него накладывается ограничение Primary Key. На поля name и date_of_creation накладываются ограничения not null.

Листинг 3.3: Создание таблицы Playlists

```
1 create table if not exists Playlists
2 (  
3     id int primary key,  
4     author_id int,  
5     name varchar(50) not null,  
6     date_of_creation date not null,  
7     description text  
8 );
```

Ниже, на листинге 3.4, представлено создание таблицы TP, которая нужна для связи таблиц Tracks и Playlists. Поле id является уникальным идентификатором связи, на него накладывается ограничение Primary Key.

Листинг 3.4: Создание таблицы TP

```
1 create table if not exists TP
2 (  
3     id int primary key,  
4     id_track int,  
5     id_playlist int  
6 );
```

Ниже, на листинге 3.5, представлено создание таблицы TU, которая нужна для связи таблиц Tracks и Users. Поле id является уникальным идентификатором связи, на него накладывается ограничение Primary Key.

Листинг 3.5: Создание таблицы TU

```
1 create table if not exists TU
2 (  
3     id int primary key,  
4     id_track int,  
5     id_user int  
6 );
```


3.3.2 Создание связей

Ниже, на листингах 3.6 - 3.8, представлено создание внешних ключей для обеспечения целостности связей между таблицами.

В таблице Playlists на поле author_id создается внешний ключ, который ссылается на поле id (первичный ключ) таблицы Users.

Листинг 3.6: Создание внешнего ключа в таблице Playlists

```
1 alter table Playlists
2   add foreign key(author_id) references Users(id);
```

В таблице TU создаются 2 внешних ключа: id_track, который ссылается на поле id (первичный ключ) таблицы Tracks, и id_playlist, который ссылается на поле id (первичный ключ) таблицы Playlists.

Листинг 3.7: Создание внешних ключей в таблице TP

```
1 alter table TP
2   add foreign key(id_track) references Tracks(id);
3
4 alter table TP
5   add foreign key(id_playlist) references Playlists(id);
```

В таблице TP создаются 2 внешних ключа: id_track, который ссылается на поле id (первичный ключ) таблицы Tracks, и id_user, который ссылается на поле id (первичный ключ) таблицы Users.

Листинг 3.8: Создание внешних ключей в таблице TU

```
1 alter table TU
2   add foreign key(id_track) references Tracks(id);
3
4 alter table TU
5   add foreign key(id_user) references Users(id);
```

3.3.3 Создание ролевой модели

Все пользователи сгруппированы на две группы для упрощения администрирования привилегий: обычные пользователи и модераторы. Также создается роль администратора.

Ниже, на листинге 3.9, представлено создание роли `ordinary_user`, которая представляет группу обычных пользователей. В соответствии с Use-Case диаграммой, представленной на рисунках 1.2 - 1.3, `ordinary_user` выдаются следующие права на таблицы БД:

- выборка, добавление, удаление данных в таблицах TP, TU, Playlists;
- выборка данных из таблиц Tracks и Users.

Листинг 3.9: Создание групповой роли обычного пользователя

```
1 create role ordinary_user;  
2  
3 grant select , insert , delete on tu to ordinary_user;  
4 grant select , insert , delete on tp to ordinary_user;  
5 grant select , insert , delete on playlists to ordinary_user;  
6 grant select on tracks to ordinary_user;  
7 grant select on users to ordinary_user;
```

Ниже, на листинге 3.10, представлено создание роли `moderator`, которая представляет группу модераторов. В соответствии с Use-Case диаграммой, представленной на рисунках 1.2 - 1.3, `moderator` выдаются следующие права на таблицы БД:

- выборка, добавление, удаление данных в таблицах TP, TU, Playlists;
- выборка, добавление, удаление, изменение (столбцы `name` и `description`) данных в таблице Tracks;
- выборка данных из таблицы Users.

Листинг 3.10: Создание групповой роли модератора

```
1 create role moderator;  
2  
3 grant select , insert , delete on tu to moderator;  
4 grant select , insert , delete on tp to moderator;
```

```

5 grant select , insert , delete on playlists to moderator;
6 grant select , insert , update (name, description),
7   delete on tracks to moderator;
8 grant select on users to moderator;

```

Ниже, на листинге 3.11, представлено создание роли administrator с параметром createrole (позволяет создавать/изменять другие роли). В соответствии с Use-Case диаграммой, представленной на рисунках 1.2 - 1.3, администратору выдаются следующие права на таблицы БД:

- выборка, добавление, удаление данных в таблицах TP, TU, Playlists;
- выборка, добавление, удаление, изменение (столбцы name и description) данных в таблице Tracks;
- выборка, добавление, удаление, изменение (столбец user_type) данных в таблице Users.

Листинг 3.11: Создание роли администратора

```

1 create role administrator with login createrole;
2
3 grant select , insert , delete on tu to administrator;
4 grant select , insert , delete on tp to administrator;
5 grant select , insert , delete on playlists to administrator;
6 grant select , insert , update (name, description),
7   delete on tracks to administrator;
8 grant select , insert , update (user_type),
9   delete on users to administrator;

```

3.3.4 Создание триггеров

Ниже, на листингах 3.12 - 3.16, представлено создание триггеров в соответствии с рисунком 2.3.

Ниже, на листинге 3.12, представлено создание триггера на добавление в таблицу Users.

Для данного триггера сначала создается триггерная функция fn_after_insert_user, которая создает роль new.login с параметрами LOGIN и PASSWORD, выдает ей членство в группе ordinary_user и возвращает переменную new.

Далее создается сам триггер `tr_after_insert_user` в режиме `after` с пометкой `for each row`, которая определяет, что триггерная функция будет срабатывать один раз для каждой строки.

Листинг 3.12: Создание триггера на вставку в Users

```
1 create or replace function fn_after_insert_user()  
2 returns trigger  
3 as  
4 $body$  
5 begin  
6     EXECUTE FORMAT( 'CREATE_ROLE_ "%I" _LOGIN_PASSWORD_%L ',  
7         cast(new.login as name),  
8         cast(new.password as varchar));  
9  
10    EXECUTE FORMAT( 'GRANT_ordinary_user_to_ "%I" ',  
11        cast(new.login as name));  
12  
13        return new;  
14 end;  
15 $body$  
16 language plpgsql;  
17  
18 create trigger tr_after_insert_user after insert on users  
19 for each row execute function fn_after_insert_user();
```

Ниже, на листинге 3.13, представлено создание триггера на удаление из таблицы Users.

Для данного триггера сначала создается триггерная функция `fn_before_delete_user`, которая

- удаляет все строки из таблиц Playlists и TU, ссылающиеся на первичный ключ (id) удаляемой из Users строки;
- удаляет роль `old.login`;
- возвращает переменную `old`.

Далее создается сам триггер `tr_before_delete_user` в режиме `before`, т.к. необходимо, чтобы он вызывался до удаления целевой строки из Users и как следствие не нарушил существующие в таблицах Playlists и TU ограничения

внешнего ключа.

Листинг 3.13: Создание триггера на удаление из Users

```
1 create or replace function fn_before_delete_user()  
2 returns trigger  
3 as  
4 $body$  
5 begin  
6         delete from tu where id_user = old.id;  
7         delete from playlists where author_id = old.id;  
8  
9         EXECUTE FORMAT( 'DROP_ROLE_%I" ',  
10             cast(old.login as name));  
11  
12         return old;  
13 end;  
14 $body$  
15 language plpgsql;  
16  
17 create trigger tr_before_delete_user after delete on users  
18 for each row execute function fn_before_delete_user();
```

Ниже, на листинге 3.14, представлено создание триггера на обновление таблицы Users.

Для данного триггера сначала создается триггерная функция fn_after_update_user, которая обновляет права доступа роли new.login:

- если поле user_type = 1, то она исключает new.login из группы moderator и выдает ему членство в ordinary_user;
- в противном случае она исключает new.login из группы ordinary_user и выдает ему членство в moderator.

Далее создается сам триггер tr_after_update_user в режиме after с пометкой for each row.

Листинг 3.14: Создание триггера на обновление Users

```
1 create or replace function fn_after_update_user()  
2 returns trigger
```

```

3  as
4  $body$
5  begin
6      IF new.user_type = 2 THEN
7          EXECUTE FORMAT( 'REVOKE_ordinary_user_from
8  ..... "%I" ', cast(new.login as name));
9
10         EXECUTE FORMAT( 'GRANT_moderator_to_ "%I" ',
11             cast(new.login as name));
12     ELSE
13         EXECUTE FORMAT( 'REVOKE_moderator_from_ "%I" ',
14             cast(new.login as name));
15
16         EXECUTE FORMAT( 'GRANT_ordinary_user_to_ "%I" ',
17             cast(new.login as name));
18     END IF;
19
20     return new;
21 end;
22 $body$
23 language plpgsql;
24
25 create trigger tr_after_update_user after update on users
26 for each row execute function fn_after_update_user();

```

Ниже, на листинге 3.15, представлено создание триггера на удаление из таблицы Tracks. Для данного триггера сначала создается триггерная функция `fn_before_delete_track`, которая удаляет все строки из таблиц TP и TU, ссылающиеся на первичный ключ (id) удаляемой из Tracks строки, и возвращает переменную `old`.

Далее создается сам триггер `tr_after_delete_track` в режиме `before`, т.к. необходимо, чтобы он вызывался до удаления целевой строки из Tracks и как следствие не нарушил существующие в таблицах TP и TU ограничения внешнего ключа. Пометка `for each row` определяет, что триггерная функция будет срабатывать один раз для каждой строки.

Листинг 3.15: Создание триггера на удаление из Tracks

```

1  create or replace function fn_before_delete_track()
2  returns trigger

```

```

3  as
4  $body$
5  begin
6      delete from tp where id_track = old.id;
7      delete from tu where id_track = old.id;
8
9      return old;
10 end;
11 $body$
12 language plpgsql;
13
14 create trigger tr_after_delete_track before delete on tracks
15 for each row execute function fn_before_delete_track();

```

Ниже, на листинге 3.16, представлено создание триггера на удаление из таблицы Playlists. Для данного триггера создается триггерная функция `fn_before_delete_playlist`, которая удаляет все строки из таблицы TP, ссылающиеся на первичный ключ (`id`) удаляемой из Playlist строки, и возвращает переменную `old`. Затем создается сам триггер `tr_before_delete_playlist` в режиме `before` с пометкой `for each row`.

Листинг 3.16: Создание триггера на удаление из Playlists

```

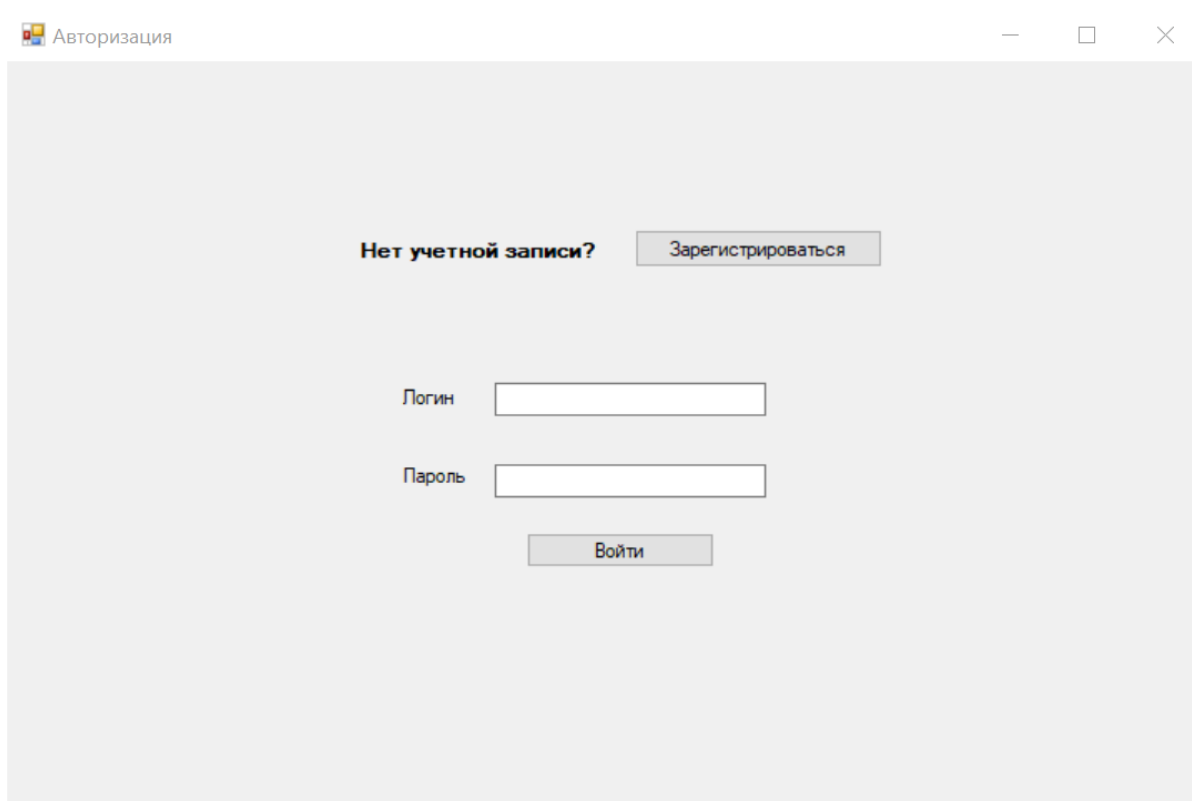
1  create or replace function fn_before_delete_playlist()
2  returns trigger
3  as
4  $body$
5  begin
6      delete from tp where id_playlist = old.id;
7
8      return old;
9  end;
10 $body$
11 language plpgsql;
12
13 create trigger tr_before_delete_playlist before delete
14 on playlists
15 for each row execute function fn_before_delete_playlist();

```

3.4 Интерфейс программы

Интерфейс программы разбит на страницы. Каждая страница предоставляет разный функционал. Ниже, на рисунках 3.2 - 3. , продемонстрирован интерфейс разработанного приложения.

На рисунке 3.2 представлена страница авторизации пользователя. Если у пользователя нет учетной записи, то ему необходимо нажать на кнопку "Зарегистрироваться" и пройти регистрацию.



Авторизация

Нет учетной записи? Зарегистрироваться

Логин

Пароль

Войти

Рис. 3.4: Страница авторизации.

На рисунке 3.3 представлена страница регистрации пользователя. После ввода всей необходимой информации пользователю нужно нажать кнопку "Зарегистрироваться" после чего он будет переведен на страницу авторизации для входа в приложение.

The image shows a registration window titled "Регистрация" (Registration). The window has a light gray background and a standard Windows-style title bar with minimize, maximize, and close buttons. The main content area is titled "Заполните поля" (Fill in the fields). It contains several input fields arranged in two columns. The left column includes fields for "Имя" (Name), "Страна" (Country), "Эл. почта" (Email), and "Дата рождения" (Date of birth). The right column includes fields for "Логин" (Login), "Пароль" (Password), and "Повторите пароль" (Repeat password). The "Дата рождения" field is a date picker showing "13 мая 2022 г.". At the bottom left, there is a blue button labeled "Зарегистрироваться" (Register).

Заполните поля			
Имя	<input type="text"/>	Логин	<input type="text"/>
Страна	<input type="text"/>	Пароль	<input type="text"/>
Эл. почта	<input type="text"/>	Повторите пароль	<input type="text"/>
Дата рождения	<input type="text" value="13 мая 2022 г."/>		
<input type="button" value="Зарегистрироваться"/>			

Рис. 3.5: Страница регистрации.

На рисунке 3.4 представлена домашняя страница, здесь пользователь может просмотреть треки/плейлисты, находящиеся в его медиатеке, получить информацию о них, добавить/удалить их из медиатеки и скачать MIDI файлы треков. Также, кликнув на кнопку "Профиль", пользователь может получить информацию о своем профиле.

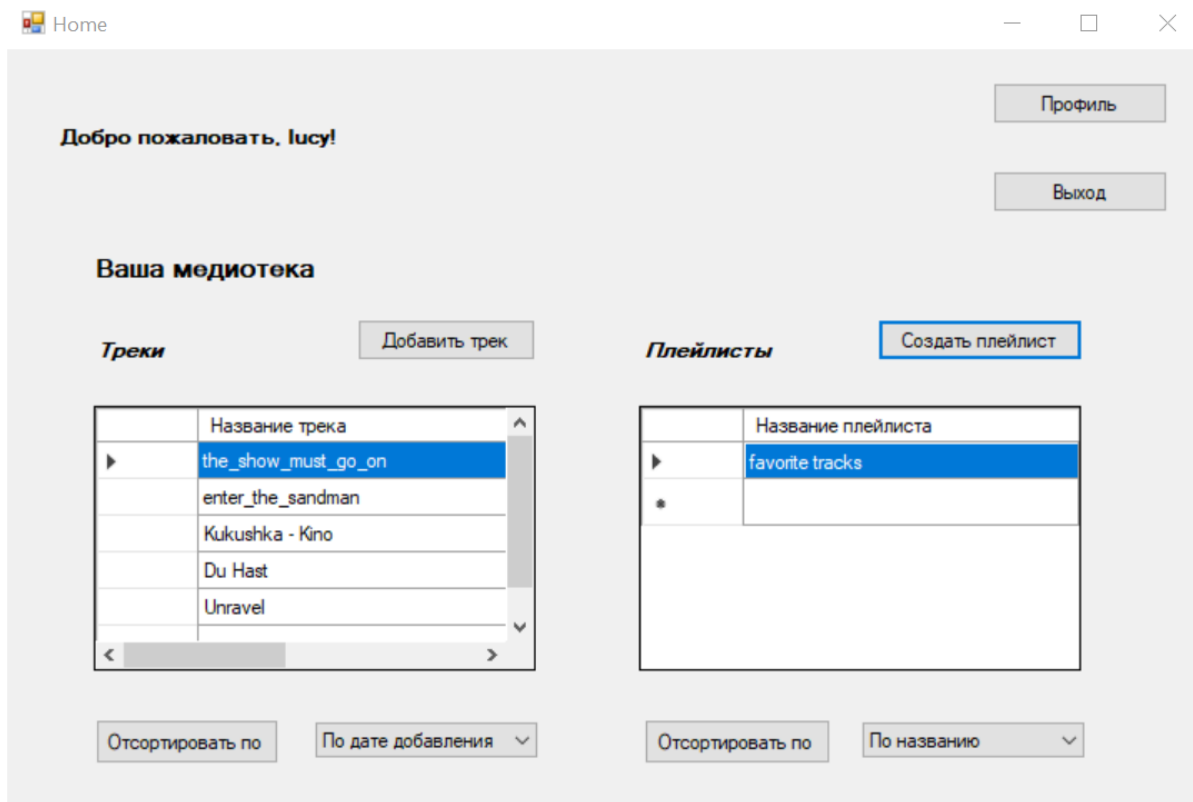


Рис. 3.6: Домашняя страница обычного пользователя.

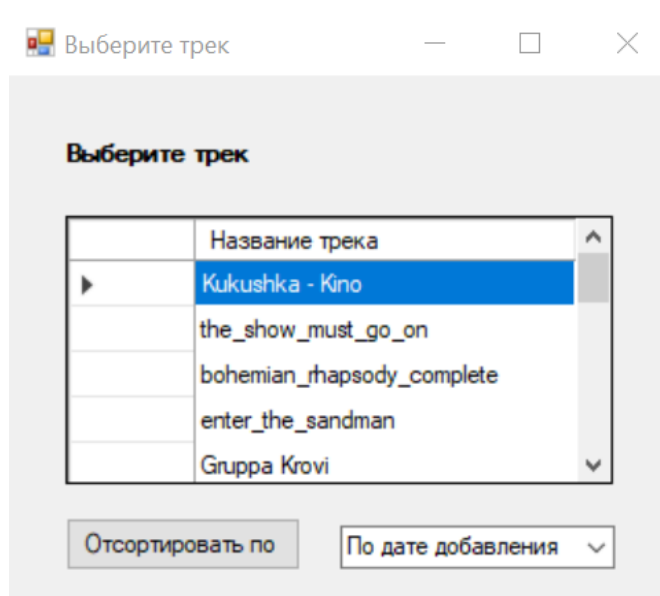


Рис. 3.7: Добавление трека в медиатеку.

О треке

Информация о треке

Скачать

Удалить

Название

Длительность

Дата создания

Описание

Рис. 3.8: Информация о треке.

О плейлисте

Информация о плейлисте

Название

Описание

Удалить плейлист

	Название трека
▶	Alejandro
	Toxicity
*	

Отсортировать по

По дате добавления ▾

Добавить трек

Рис. 3.9: Информация о плейлисте.

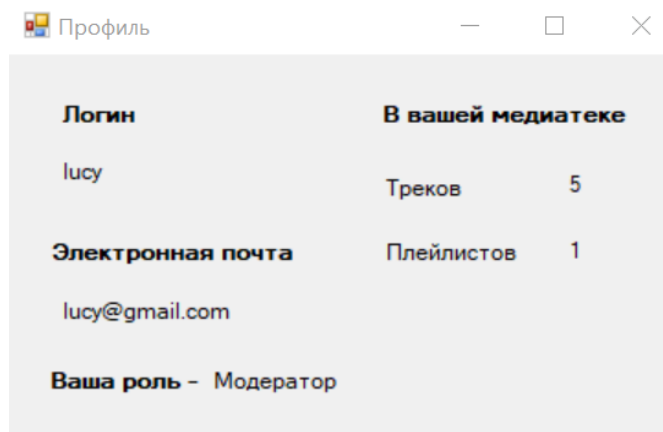


Рис. 3.10: Информация о профиле.

Для модератора доступен дополнительный функционал. При нажатии на кнопку "Библиотека треков" модератор может посмотреть список треков, из которых состоит библиотека, добавить/удалить треки и отредактировать информацию о треке (название/описание).

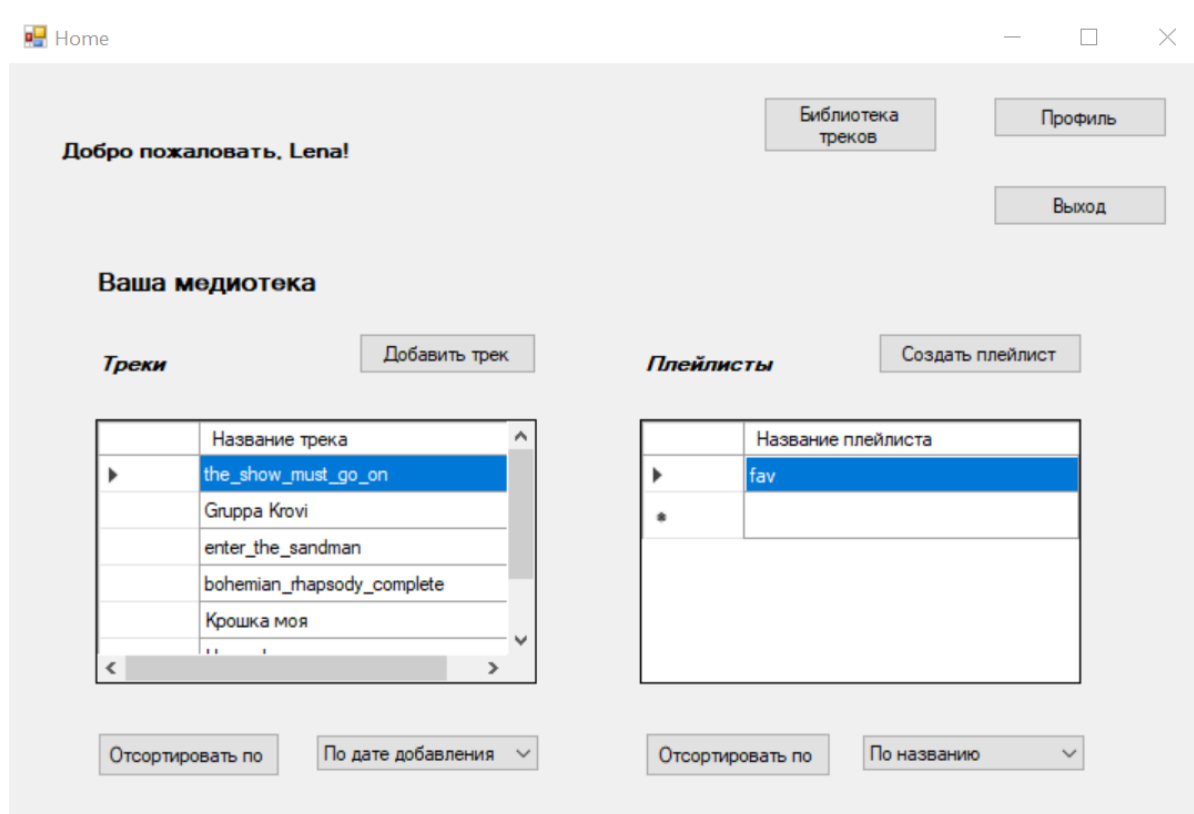


Рис. 3.11: Домашняя страница модератора.

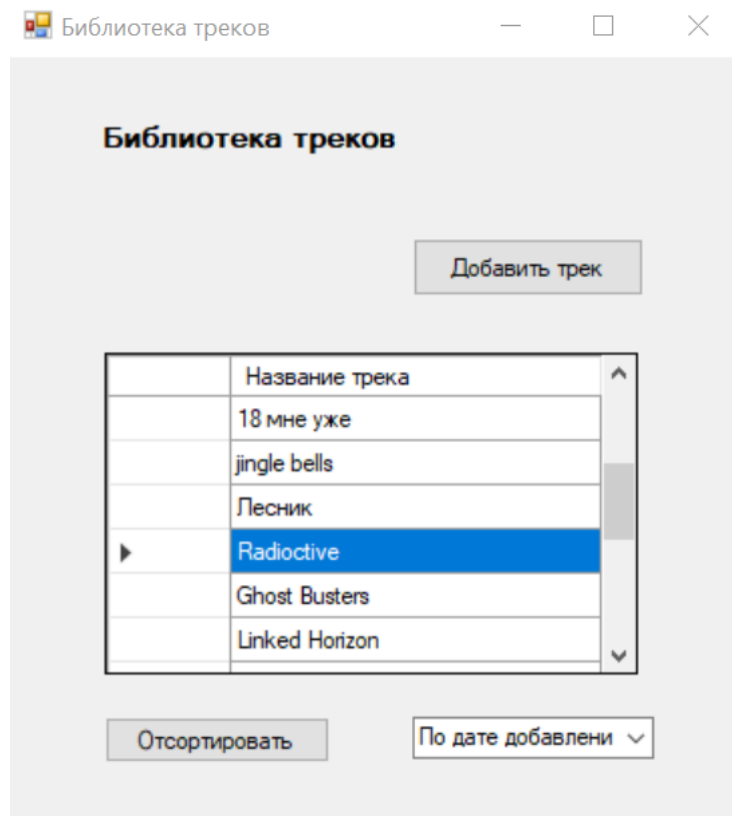


Рис. 3.12: Библиотека MIDI треков.

Также при нажатии на кнопку "Добавить трек" модератор может добавить новый трек в "библиотеку MIDI-треков".

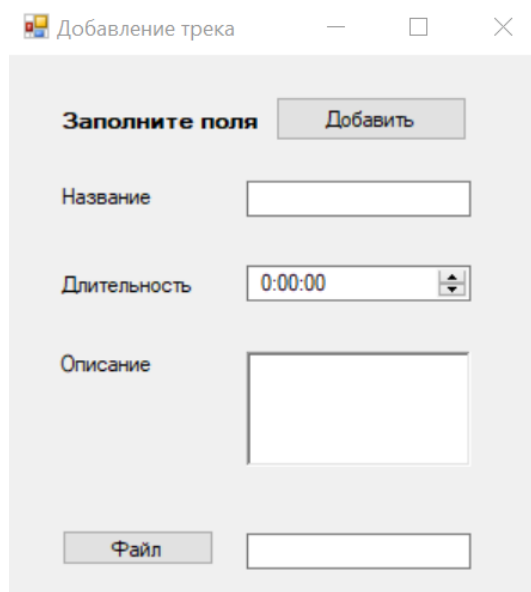


Рис. 3.13: Добавление нового трека в библиотеку.

О треке

Информация о треке

Скачать

Удалить

Сохранить изменения

Название: Radioactive

Длительность: 00:05:34

Дата создания: 14.05.2022

Описание: very popular song by Imagine Dragons

Рис. 3.14: Информация о треке библиотеки.

На домашней странице администратора есть кнопка "Пользователи" при нажатии на которую администратор может просмотреть список зарегистрированных в системе пользователей и изменить их роли (предоставить/отобрать права модератора).

Домой

Добро пожаловать, Elena!

Библиотека треков

Профиль

Пользователи

Выход

Ваша медиотека

Треки Добавить трек

	Название трека
▶	the_show_must_go_on
	Имперский марш
	Seven Nation Army
	Blue Bird
	Лунная соната

Отсортировать по По дате добавления

Плейлисты Создать плейлист

	Название плейлиста
▶	various tracks
*	

Отсортировать по По названию

Рис. 3.15: Домашняя страница администратора.

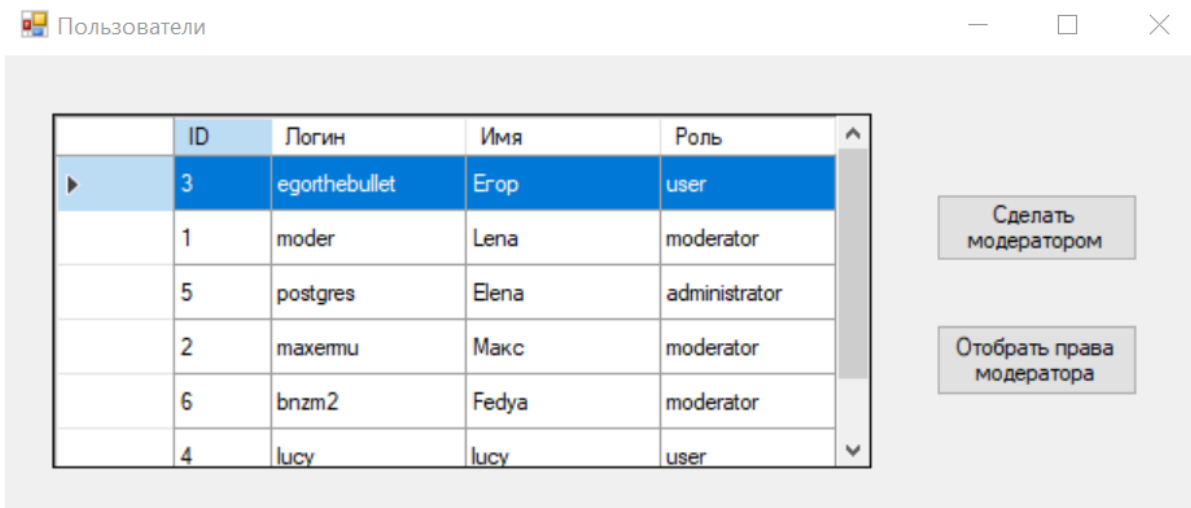


Рис. 3.16: Информация о пользователях.

Вывод

В данном разделе были выбраны средства реализации: язык программирования C# и среда разработки VisualStudio, для создания пользовательского интерфейса использовалась библиотека WindowsForms. Также были рассмотрены структура и состав реализованных классов, составлена UML-диаграмма классов, продемонстрирован разработанный интерфейс, предоставлены листинги создания объектов БД и ролевой модели.

4 | Экспериментальный раздел

В данном разделе будет проведен анализ времени и стоимости выполнения запросов к БД с использованием индексов и без.

4.1 Сравнение производительности

В данной работе используются следующие виды запросов: поиск по первичному ключу, поиск по полю с фильтрацией, объединение таблиц с поиском по внешнему ключу. Сравнение производительности проводилось на таблицах из 1000 строк.

Для повышения производительности в данной работе будут использоваться В-tree индексы, так как с помощью В-дерева можно проиндексировать любые данные, которые могут быть отсортированы, т. е. для которых применимы операции сравнения больше/меньше/равно. Также В-дерево позволяет ускорить практически любой запрос, условие которого является выражением, состоящим из полей входящих в индекс, логических операторов и операций равенства/сравнения.

4.1.1 Поиск по первичному ключу

Один кластеризованный индекс для таблицы создаётся неявно — это первичный ключ.

Выполним поиск конкретного пользователя в таблице Users по первичному ключу (Id):

```
1 select name, login from users where id = 342;
```



```
db_cp=# explain (analyze) select name, login from users where id = 521;
                                QUERY PLAN
-----
Index Scan using users_pkey on users  (cost=0.28..8.29 rows=1 width=22) (actual time=0.017..0.018 rows=1 loops=1)
  Index Cond: (id = 521)
  Planning Time: 4.319 ms
  Execution Time: 0.029 ms
(4 строки)

Время: 4,853 мс
```

Рис. 4.1: Результат выполнения запроса.

Время, затраченное на поиск: 4,853 мс.

Повторный запрос затратил 0,430 мс, а третий — 0,428 мс.

Как уже было сказано выше, столбцы первичного ключа изначально имеют индекс (в плане запроса используется индексное сканирование Index Scan), поэтому повысить производительность здесь скорее всего не получится. Исходя из результатов можно сказать, что повторные запросы с теми же параметрами увеличивают скорость поиска.

4.1.2 Поиск по полю с фильтрацией

Выполним поиск всех плейлистов из таблицы Playlists, которые были созданы определенным пользователем (`author_id = 521`):

```
1 select id, name from playlists where author_id = 521;
```

```
db_cp=# explain (analyze, timing on) select id, name from playlists where author_id = 521;
                                QUERY PLAN
-----
Seq Scan on playlists  (cost=0.00..23.54 rows=2 width=10) (actual time=0.077..0.154 rows=2 loops=1)
  Filter: (author_id = 521)
  Rows Removed by Filter: 1001
  Planning Time: 0.280 ms
  Execution Time: 0.164 ms
(5 строк)

Время: 0,847 мс
```

Рис. 4.2: Результат выполнения запроса.

Без индекса такой запрос выполняется за 0,847 мс из 1000 строк. Затраты выполнения запроса (costs) равны 23,54. Также в данном запросе используется Seq Scan — последовательное, блок за блоком, чтение данных таблицы. Т.е, в отсутствии индекса БД придется просмотреть каждую строку таблицы до тех

пор, пока она не найдет нужную.

Создадим индекс для поля `author_id` таблицы `Playlists`:

```
1 create index author_id_idx on playlists
2 using btree(author_id);
```

Выполнив предыдущий запрос повторно, получается следующий результат:

```
db_cp=# explain (analyze, timing on) select id, name from playlists where author_id = 521;
                                         QUERY PLAN
-----
Bitmap Heap Scan on playlists  (cost=4.29..9.76 rows=2 width=10) (actual time=0.046..0.048 rows=2 loops=1)
  Recheck Cond: (author_id = 521)
  Heap Blocks: exact=2
-> Bitmap Index Scan on author_id_idx  (cost=0.00..4.29 rows=2 width=0) (actual time=0.041..0.041 rows=2 loops=1)
    Index Cond: (author_id = 521)
Planning Time: 0.066 ms
Execution Time: 0.066 ms
(7 строк)

Время: 0,497 мс
```

Рис. 4.3: Результат выполнения запроса.

Время выполнения запроса уменьшилось в два раза и составило 0,497 мс, а затраты снились с 23,54 до 9,76 (уменьшились в 2,4 раза), что говорит о значительном повышении производительности.

4.1.3 Объединение таблиц с поиском по внешнему ключу

Выполним поиск информации о всех треках, добавленных в определенный плейлист (`id_playlist = 82`):

```
1 select *
2 from tp join tracks on tp.id_track = tracks.id
3 where tp.id_playlist = 82;
```

```

db_cp=# explain (analyze) select * from tp join tracks on tp.id_track = tracks.id where tp.id_playlist = 82;
                                QUERY PLAN
-----
Hash Join  (cost=18.54..25.78 rows=2 width=105) (actual time=0.098..0.118 rows=2 loops=1)
  Hash Cond: (tracks.id = tp.id_track)
    -> Seq Scan on tracks  (cost=0.00..6.15 rows=215 width=93) (actual time=0.010..0.022 rows=215 loops=1)
    -> Hash  (cost=18.51..18.51 rows=2 width=12) (actual time=0.077..0.077 rows=2 loops=1)
          Buckets: 1024  Batches: 1  Memory Usage: 9kB
          -> Seq Scan on tp  (cost=0.00..18.51 rows=2 width=12) (actual time=0.010..0.075 rows=2 loops=1)
                Filter: (id_playlist = 82)
                Rows Removed by Filter: 999
Planning Time: 0.183 ms
Execution Time: 0.135 ms
(10 строк)

Время: 0,683 мс

```

Рис. 4.4: Результат выполнения запроса.

Время выполнения запроса - 0,683 мс, затраты на выполнение запроса - 25,75.

Создадим индекс для внешнего ключа (`id_playlist`) таблицы `Тр` и выполним запрос повторно:

```

1 create index id_playlist_idx on tp using btree(id_playlist);

```

```

db_cp=# explain (analyze) select * from tp inner join tracks on tp.id_track = tracks.id where tp.id_playlist = 82;
                                QUERY PLAN
-----
Hash Join  (cost=8.88..16.12 rows=2 width=105) (actual time=0.035..0.057 rows=2 loops=1)
  Hash Cond: (tracks.id = tp.id_track)
    -> Seq Scan on tracks  (cost=0.00..6.15 rows=215 width=93) (actual time=0.008..0.022 rows=215 loops=1)
    -> Hash  (cost=8.85..8.85 rows=2 width=12) (actual time=0.016..0.016 rows=2 loops=1)
          Buckets: 1024  Batches: 1  Memory Usage: 9kB
          -> Bitmap Heap Scan on tp  (cost=4.29..8.85 rows=2 width=12) (actual time=0.011..0.014 rows=2 loops=1)
                Recheck Cond: (id_playlist = 82)
                Heap Blocks: exact=2
                -> Bitmap Index Scan on id_playlist_idx  (cost=0.00..4.29 rows=2 width=0) (actual time=0.008..0.008 rows=2 loops=1)
                      Index Cond: (id_playlist = 82)
Planning Time: 0.168 ms
Execution Time: 0.081 ms
(12 строк)

Время: 0,673 мс

```

Рис. 4.5: Результат выполнения запроса.

Время выполнения запроса практически не изменилось и составило 0,673 мс, но затраты уменьшились в 1.6 раза и составили 16,12.

Вывод

Проанализировав полученные результаты, можно сказать, что индексирование значительно ускоряет `SELECT` запросы, а также они не влияют на скорость

вставки данных, следовательно использование индексов в данной работе повысит ее производительность.

Заключение

Во время выполнения курсового проекта были рассмотрены существующие виды БД, описана структура базы данных и приложения, разработано программное обеспечение, предоставляющее интерфейс к базе данных.

Программа "Библиотека MIDI треков" реализована таким образом, что пользователь может добавлять треки в свою медиотеку, просматривать информацию о них, создавать из них плейлисты; модератор также может добавлять новые треки в "библиотеку MIDI треков" и изменять информацию о них (название/описание); администратор дополнительно может изменять права пользователей.

В ходе выполнения поставленной задачи были изучены возможности языка C# и библиотеки WindowsForms. Получен опыт работы с СУБД PostgreSQL и системой управления БД pgAdmin, получены знания в области баз данных.

Цель работы достигнута, выполнены следующие задачи:

1. формализовано задание и определен необходимый функционал;
2. описана структура базы данных, включая объекты, из которых она состоит, и связи между ними;
3. создана и заполнена БД;
4. разработано ПО, предоставляющее интерфейс для доступа к БД;
5. исследовано время и стоимость выполнения запросов к БД с использованием индексов и без.

В качестве дальнейшего развития проекта можно предложить добавление в приложение возможности генерации трека в заданном стиле/темпе или на основе биологических триггеров человека (пульс, дыхание) и воспроизведение сгенерированного трека.

Список литературы

1. MidiStock. [Электронный ресурс]. Режим доступа: <https://midistock.ru/> (дата обращения: 28.05.2022)
2. BitMidi. [Электронный ресурс]. Режим доступа: <https://bitmidi.com/> (дата обращения: 28.05.2022)
3. Midiworld. [Электронный ресурс]. Режим доступа: <https://www.midiworld.com/> (дата обращения: 28.05.2022)
4. Кузнецов С. Д. Основы современных баз данных // М: Центр Информационных Технологий. – 1998.
5. Парфенов Ю. П. Постреляционные хранилища данных: учебное пособие. – 2016.
6. C#: Документация. [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 16.04.2022)
7. Visual Studio: Документация. [Электронный ресурс]. Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2022> (дата обращения: 19.04.2022)
8. Информационные технологии интеллектуальной поддержки принятия решений (ITIDS'2018) : Труды VI Всероссийской конференции (с приглашением зарубежных ученых), Уфа-Ставрополь, 28–31 мая 2018 года. – Уфа-Ставрополь: ГОУ ВПО "Уфимский государственный авиационный технический университет 2018. – 303 с. – ISBN 978-5-4221-1120-6. – EDN YLUBMT.
9. PostgreSQL: Документация. [Электронный ресурс]. Режим доступа: <https://postgrespro.ru/docs/postgresql/> (дата обращения: 22.04.2022).