МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ  
Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ   
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра информационных систем и технологий

Специальность 1-40 05 01 «Информационные системы и технологии»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА КУРСОВОГО ПРОЕКТА**

по дисциплине «Базы данных»

Тема «Реализация базы данных стримингового сервиса для прослушивания музыки на PostgreSQL с использованием технологии репликация»

**Исполнитель**

студент 2 курса 1 группы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ И. О. Курильчик

(подпись, дата)

**Руководитель**

Доцент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. А. Блинова

(должность, уч. звание) (подпись, дата)

Допущен(а) к защите \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

дата, подпись

Курсовой проект защищен с оценкой

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. А. Блинова

подпись дата инициалы и фамилия

Минск 2023

**Содержание**

[Введение 3](#_Toc135296740)

[1 Анализ требований к программному средству 4](#_Toc135296741)

[1.1 Аналитический обзор аналогов 4](#_Toc135296742)

[1.1.1 Аналог Spotify 4](#_Toc135296743)

[1.1.2 Аналог SoundCloud 6](#_Toc135296744)

[1.2 Разработка функциональных требований, определение вариантов использования 7](#_Toc135296745)

[1.3 Вывод 10](#_Toc135296746)

[2 Разработка архитектуры проекта 11](#_Toc135296747)

[2.1 Диаграмма базы данных, взаимосвязь всех компонентов. 11](#_Toc135296748)

[3 Разработка модели базы данных 14](#_Toc135296749)

[3.1 Создание необходимых объектов 14](#_Toc135296750)

[3.1.2 Индексы базы данных 15](#_Toc135296751)

[3.2 Описание используемой технологии 16](#_Toc135296752)

[3.3 Вывод 18](#_Toc135296753)

[4 Установка, настройка и использование PosgtreSQL 15 19](#_Toc135296754)

[4.1 Установка PostgreSQL 19](#_Toc135296755)

[4.2 Создание таблиц 19](#_Toc135296756)

[4.3 Создание ролей для разграничения 20](#_Toc135296757)

[4.4 Выборка данных из таблиц 22](#_Toc135296758)

[4.4.1 Заполнение таблиц 100 000 строк 23](#_Toc135296759)

[4.4.2 Добавление данных в таблицы 24](#_Toc135296760)

[4.4.3 Удаление данных из таблиц 25](#_Toc135296761)

[4.4.4 Изменение данных в таблицах 25](#_Toc135296762)

[4.5 Описание процедур экспорта и импорта 26](#_Toc135296763)

[4.6 Вывод 28](#_Toc135296764)

[5 Тестирование 29](#_Toc135296765)

[5.1 Тестирование производительности базы данных 29](#_Toc135296766)

[5.2 Вывод 30](#_Toc135296767)

[Заключение 31](#_Toc135296768)

[Список литературных источников 32](#_Toc135296769)

[Приложение А 33](#_Toc135296770)

# Введение

Для данного проекта мы ставим перед собой задачу разработки реляционной базы данных для нашей музыкальной платформы, которая позволит нашим пользователям получить доступ к множеству музыкальных произведений, доступных на платформе.

База данных представляет собой организованную коллекцию данных, которая хранится на компьютерной системе. Она служит для хранения, организации и управления большим объемом структурированных данных. Наиболее распространенной формой организации данных является таблица, состоящая из столбцов и строк. Каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись.

В нашей работе мы выбрали СУБД PostgreSQL, так как она обладает высокой надежностью и производительностью, что обеспечивает эффективное хранение и обработку данных.

Одной из ключевых технологий, применяемых в нашей базе данных, является репликация. Она позволяет нам дублировать все данные на второй сервер, что обеспечивает возможность восстановления данных в случае необходимости отката или после несанкционированного доступа. Применение данной технологии гарантирует стабильную работу базы данных и быстрое восстановление данных при необходимости.

В нашем проекте мы определяем основные требования к нашему приложению:

* Реализация ролей администратора и пользователей, чтобы обеспечить различные уровни доступа и функциональность.
* Возможность поиска аудиозаписей по альбому, исполнителю или жанру, чтобы пользователи могли быстро найти интересующие их треки.
* Возможность загрузки аудиозаписей на платформу пользователями, чтобы расширить музыкальную библиотеку и предоставить возможность делиться с другими пользователями.
* Взаимодействие с базой данных при помощи хранимых процедур и функций, чтобы обеспечить эффективную обработку данных и выполнение операций в базе данных.

В пояснительной записке будет содержаться информация о существующих аналогичных продуктах, их структуре и реализации. Также будет представлена схема базы данных и объяснение использования различных приемов при ее создании.

1. Анализ требований к программному средству

## Аналитический обзор аналогов

Музыка представляет собой одну из самых известных и широко распространенных форм искусства, обладающую богатой историей. Она оказывает значительное влияние на культуру в целом и вливается в наше общество. В настоящее время музыкальные платформы играют важную роль на музыкальном рынке, предоставляя людям доступ к огромной библиотеке музыкальных произведений различных стилей и эпох.

Одним из основных преимуществ музыкальных платформ является возможность получить доступ к музыке в любое время и в любом месте с помощью смартфонов, компьютеров и других устройств с доступом к Интернету. Это делает музыкальные платформы известными и популярными среди миллионов пользователей по всему миру.

Вышеприведённые факты свидетельствуют о том, что в сфере музыкальных стриминговых сервисов присутствует большая конкуренция. Каждый разработчик желает предоставить пользователю удобное и доступное приложение, которое будет выполнять запросы к базе данных, размеры которой могут превышать десятки миллионов записей, поэтому перед разработчиками стоит задача разработать структуру, которая будет выдерживать огромные нагрузки и хранить личные данные пользователей под защитой от злоумышленников.

### 1.1.1 Аналог Spotify

Spotify [1] - это известная и популярная музыкальная платформа, предоставляющая пользователям возможность стриминга музыки. Она позволяет создавать плейлисты, делиться ими с друзьями, слушать радио и находить новую музыку, основываясь на индивидуальных предпочтениях.

Spotify - это музыкальная платформа, которая обладает широким спектром функций для удобного и насыщенного музыкального опыта пользователей. Одной из ключевых возможностей Spotify является простой и эффективный поиск музыки. Пользователи могут легко находить треки, альбомы, исполнителей и жанры, используя удобные инструменты поиска по платформе. Это позволяет пользователям найти как уже знакомые им музыкальные произведения, так и открыть для себя новую музыку в соответствии с их вкусами и предпочтениями.

Кроме того, Spotify предлагает персонализированные рекомендации, основанные на музыкальных предпочтениях каждого пользователя. Платформа анализирует историю прослушивания, предпочтения и поведение пользователя, чтобы создавать индивидуальные рекомендации, помогая пользователю открыть новые песни, артистов и жанры, которые ему могут понравиться. Кроме того, Spotify предлагает широкий выбор плейлистов, созданных как самими пользователями, так и профессиональными кураторами платформы. Это дает возможность обнаруживать новые треки и настроение, а также делиться своими музыкальными открытиями с друзьями.

На рисунке 1.1 представлен пример интерфейса этого сервиса.

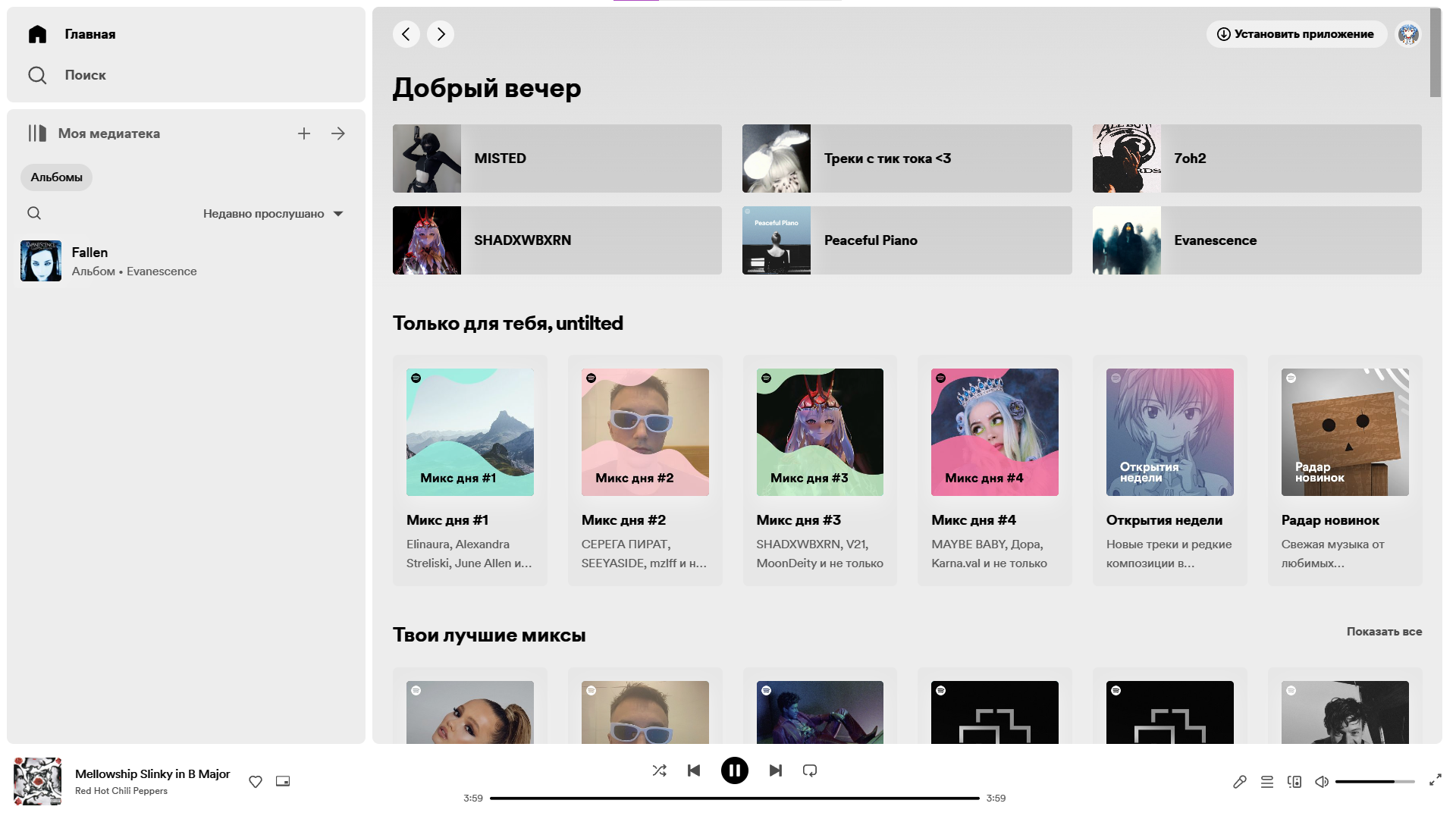


Рисунок 1.1 – Интерфейс и функционал сервиса Spotify

В Spotify пользователи имеют возможность создавать собственные плейлисты, составлять их из своих любимых треков и делиться ими с другими пользователями. Кроме того, Spotify позволяет сохранять музыку для офлайн-воспроизведения, чтобы пользователи могли слушать свои любимые треки в любое время и в любом месте, даже без доступа к Интернету. Платформа также предоставляет доступ к широкому выбору радиостанций и подкастов, включая оригинальный контент, созданный самим Spotify и другими контентными партнерами.

Spotify предлагает два варианта подписки: бесплатную и платную. Бесплатная версия содержит рекламные вставки и имеет некоторые ограничения, включая случайный порядок воспроизведения и ограничение на пропуск треков. Платная версия Spotify позволяет получить полный и неограниченный доступ к музыкальной библиотеке, отсутствие рекламы и дополнительные функции, такие как возможность скачивать музыку и прослушивать ее офлайн, улучшенное качество звука и возможность пропускать треки без ограничений. Подписка на платную версию также предоставляет доступ к эксклюзивному контенту и раннему доступу к новым релизам.

### 1.1.2 Аналог SoundCloud

SoundCloud [2] - это популярная музыкальная платформа, которая предлагает уникальные возможности для музыкантов и меломанов. SoundCloud позволяет пользователям не только стримить музыку онлайн, но и делиться своими собственными аудиозаписями с миром.

Одним из ключевых преимуществ SoundCloud является его открытая и коллаборативная природа. Пользователи могут загружать свои музыкальные треки и подкасты на платформу, давая возможность другим слушателям оценить их творчество.

SoundCloud предлагает богатую библиотеку музыкального контента, включая треки различных жанров, ремиксы, микстейпы и многое другое. Пользователи могут легко искать музыку по артистам, песням или жанрам, а также открывать новые таланты и следить за своими любимыми музыкантами.

В дополнение к прослушиванию музыки, SoundCloud предлагает также широкий выбор подкастов, аудиокниг и радиошоу. Это позволяет пользователям расширить свой музыкальный опыт, открыть новые и интересные форматы аудиоконтента и узнать больше о любимых артистах и жанрах.

Интерфейс SoundCloud интуитивно понятен и удобен для использования. Пользователи могут легко найти интересующие их аудиозаписи, следить за активностью других пользователей и создавать свои плейлисты. SoundCloud также предоставляет возможность скачивать музыку для офлайн прослушивания, что позволяет наслаждаться любимыми треками даже без доступа к Интернету.

Пример интерфейса данного сервиса представлен на рисунке 1.2.

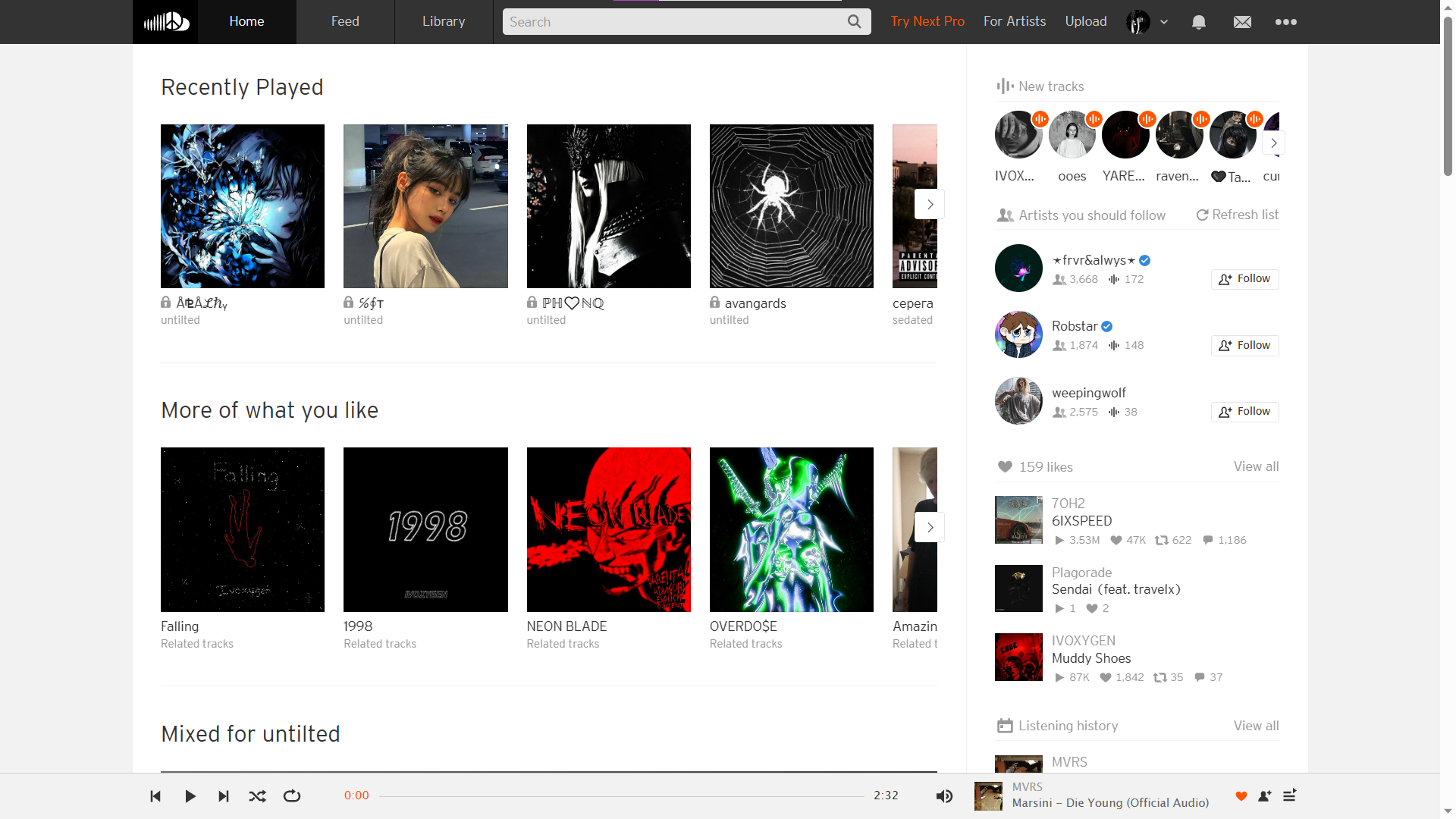


Рисунок 1.2 – Интерфейс и функционал сервиса SoundCloud

В данном разделе были проведены аналитические обзоры двух популярных музыкальных сервисов - Spotify и SoundCloud. Оба сервиса представляют собой уникальные платформы, предоставляющие пользователю возможность наслаждаться музыкой онлайн и настраивать свой музыкальный опыт в соответствии с личными предпочтениями.

Spotify - это узнаваемая и широко используемая платформа, которая предлагает огромную библиотеку музыкального контента, включая треки различных жанров и эпох. С помощью Spotify пользователи могут не только искать и слушать свои любимые песни и исполнителей, но и создавать собственные плейлисты, делиться ими с друзьями и получать персонализированные рекомендации, основанные на их музыкальных предпочтениях. Кроме того, Spotify предлагает удобные функции офлайн-воспроизведения и взаимодействия с другими пользователями, делая его одним из лидеров в сфере потоковой музыки.

SoundCloud, с другой стороны, является более коллаборативной платформой, которая позволяет музыкантам и аудио-творцам загружать, распространять и продвигать свою собственную музыку. SoundCloud отличается от Spotify тем, что здесь можно найти множество новых и независимых артистов, ремиксы, микстейпы и другие эксклюзивные аудиозаписи, которые не всегда доступны на других музыкальных платформах. SoundCloud также предоставляет социальные функции, позволяющие пользователям взаимодействовать друг с другом, обмениваться мнениями и создавать сообщества вокруг музыкальных интересов.

## 1.2 Разработка функциональных требований, определение вариантов использования

Функциональные требования базы данных определяют, как база данных должна обрабатывать данные и предоставлять пользовательскому интерфейсу необходимую функциональность. Они включают описание способа хранения и организации данных, механизмы поиска и выборки данных, обновления данных и защиты данных. Также функциональные требования могут включать интеграцию базы данных с другими системами и программами.

Для музыкальной платформы, функциональные требования могут включать следующие возможности:

* Хранение информации о музыкальных треках и пользователях: база данных должна иметь возможность хранить и организовывать информацию о треках, включая название, исполнителя, жанр, дату публикации и другие метаданные.
* Поиск музыки по категориям и критериям: пользователи должны иметь возможность искать музыку по различным категориям, таким как жанр или исполнитель.
* Создание и управление плейлистами: пользователи должны иметь возможность создавать свои собственные плейлисты, добавлять в них треки и управлять содержимым плейлистов.
* Оценка и прослушивание музыки: пользователи должны иметь возможность добавлять треки и альбомы в свою медиатеку, также иметь возможность подписываться на исполнителей.

Помимо функциональных требований, важно определить роли пользователей и их варианты использования системы. Роли пользователей определяют набор прав и доступа, которые пользователь получает в системе. В данном проекте предусмотрены следующие роли пользователей:

* Исполнитель: Эта роль относится к музыкантам или аудио-творцам, которые могут загружать свою собственную музыку на платформу, управлять своими аудиозаписями, публиковать и изменять альбомы.
* Слушатель: Эта роль относится к пользователям, которые могут прослушивать музыку, создавать плейлисты, добавлять треки и альбомы в избранное, подписываться на исполнителей, а также оформлять преимум версию подписки.
* Администратор: Эта роль относится к администраторам платформы, которые имеют расширенные права доступа, могут управлять пользователями, модерировать контент, удалять и создавать пользователей, добавлять новые жанры и изменять все пользовательские данные.

Варианты использования системы представлены на рисунке 1.3 и описывают, как каждая роль будет взаимодействовать с системой в соответствии со своими правами и функциональными возможностями. Это позволяет определить, какие функции доступны для каждой роли, какие данные доступны и как организована навигация в системе. Варианты использования обычно представляются в виде UML-диаграмм для наглядного отображения взаимодействия между пользователями и системой.

В начале работы с приложением, пользователь является гостем, что означает, что у него есть доступ только для просмотра информации без возможности прослушивания музыки. Однако, после регистрации, пользователь становится Слушателем.

В роли Слушателя пользователь получает широкий спектр возможностей. Он может прослушивать музыку, добавлять понравившиеся треки в свой персональный плейлист, оценивать музыкальные композиции и осуществлять поиск музыки по категориям и критериям. Кроме того, Слушатель может создавать собственную музыкальную библиотеку, где он может хранить любимые треки в соответствии с настроением или жанром.

Роль Администратора заключается в управлении музыкальной платформой и мониторинге действий пользователей. Администратор имеет полный доступ к просмотру информации о всех пользователях, включая их действия на платформе. При необходимости, администратор может редактировать информацию о треках и пользователях. Например, он может изменять жанры у треков, а также изменять личную информацию о пользователях.

Роль Исполнителя на музыкальной платформе представляет собой особую категорию пользователей, которые являются создателями и представителями музыкального контента. Исполнитель играет важную роль в экосистеме платформы, предоставляя свои музыкальные композиции и альбомы для прослушивания и взаимодействия с аудиторией.

Исполнитель имеет возможность загружать свои музыкальные треки на платформу и делиться ими со своей аудиторией. Он может создавать свой профиль, на котором отображаются его музыкальные работы.

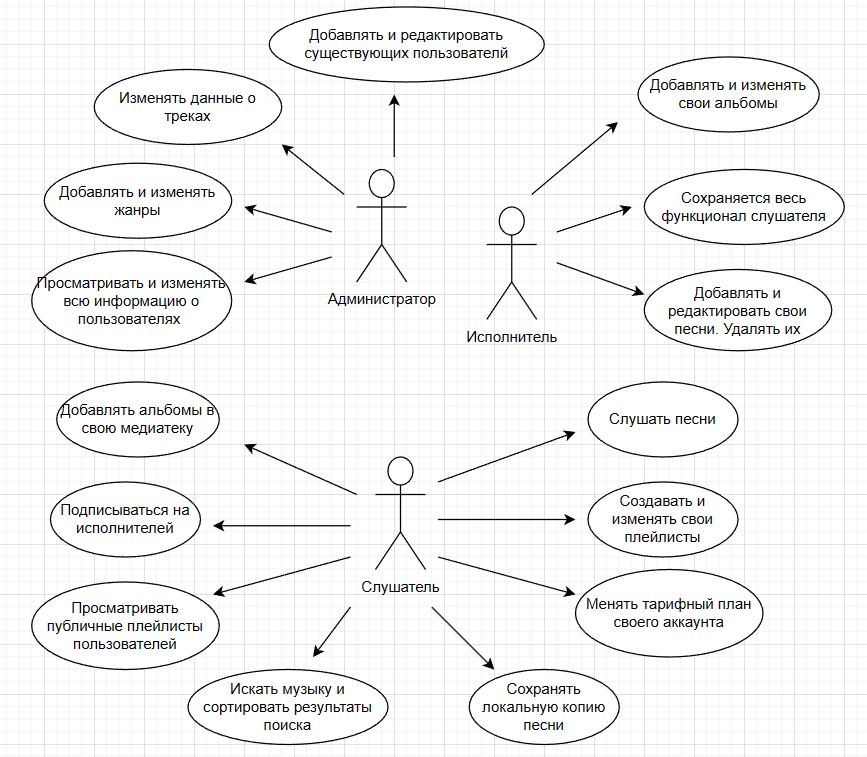


Рисунок 1.3 – UML диаграмма вариантов использования

В данном разделе были представлены роли и функциональные возможности пользователей на музыкальной платформе. Начиная с роли гостя, который имеет только ограниченный доступ к информации, пользователь после регистрации становится Слушателем и получает широкий спектр возможностей, таких как прослушивание музыки, создание персональных плейлистов, поиск музыки и альбомов по различным критериям.

Администратор, в свою очередь, играет важную роль в управлении и мониторинге платформы, имея полный доступ к информации о пользователях и треках. Это позволяет администратору контролировать и редактировать данные, включая категории, жанры и личную информацию.

Особую категорию пользователей представляют Исполнители, которые создают и представляют собственный музыкальный контент. Исполнители могут загружать свои треки и альбомы на платформу, изменять существующие музыкальные материал.

## 1.3 Вывод

В результате проведенного аналитического обзора музыкальных платформ и сервисов, существующих на рынке, мы определили ключевые характеристики и функциональные возможности, которые необходимо учесть при разработке нашей системы. Также мы выявили функциональные требования к базе данных и определили роли пользователей, а также их варианты использования в системе.

На основе этой информации мы разработали UML-диаграмму, которая наглядно отображает основные функции, доступные для каждой роли пользователей. Это поможет нам лучше понять взаимодействие между пользователями и системой, а также определить необходимые функциональные возможности для каждой роли.

В целом, проведенный обзор и разработка UML-диаграммы позволили нам получить полное представление о требованиях и функционале, которые будут включены в разрабатываемую систему. Это служит важным руководством при создании надежной и удобной музыкальной платформы, учитывая опыт и успешные решения, представленные на рынке.

1. Разработка архитектуры проекта

## 2.1 Диаграмма базы данных, взаимосвязь всех компонентов.

Диаграмма таблиц базы данных (Database Table Diagram) представляет собой визуальное отображение структуры базы данных и связей между таблицами, которые хранятся в этой базе данных. Данная диаграмма позволяет наглядно представить схему базы данных и взаимосвязи между различными таблицами.

На рисунке 2.1 представлена диаграмма базы данных, которая демонстрирует организацию таблиц и их взаимосвязи в рамках данной системы.

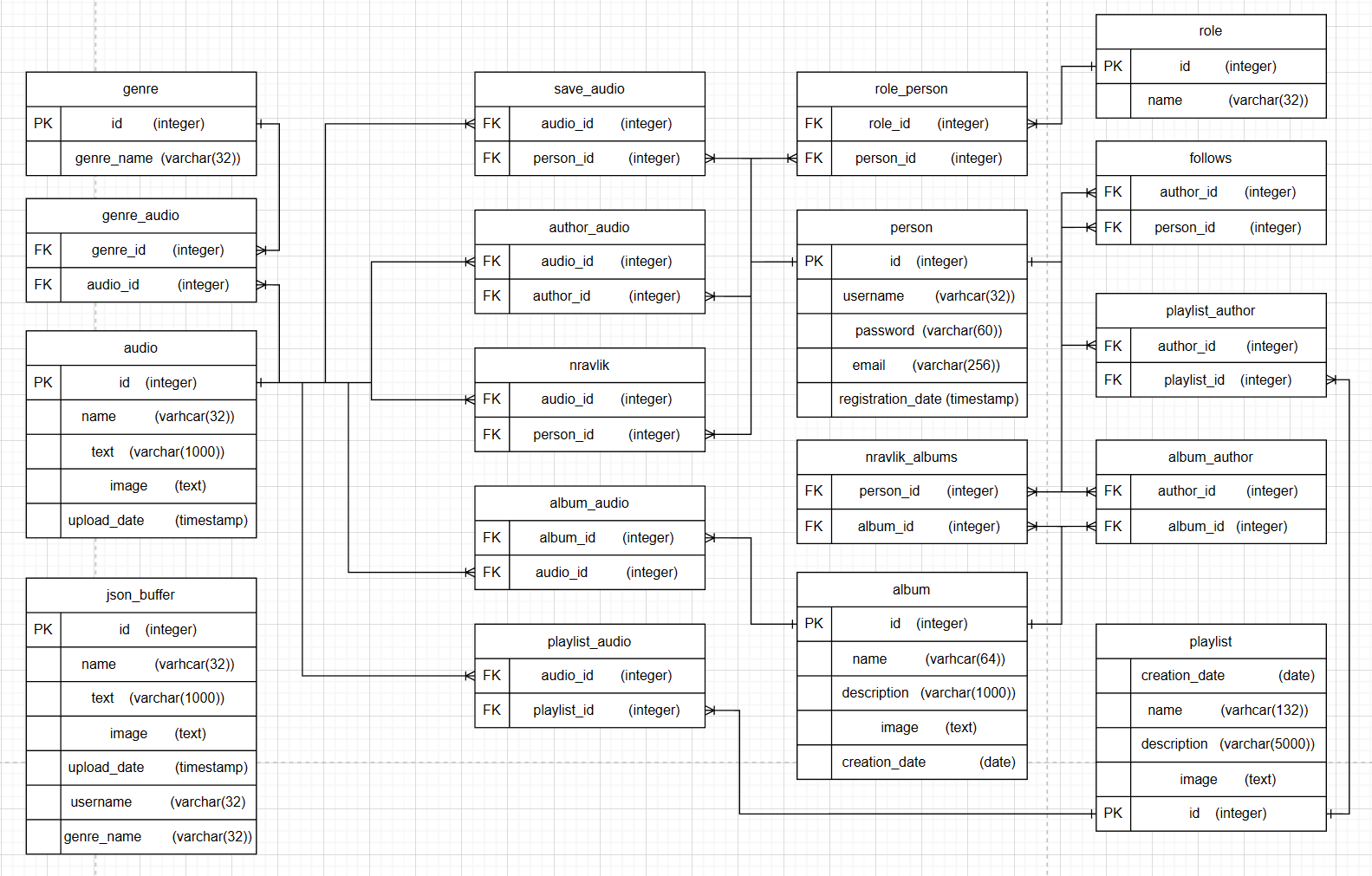


Рисунок 2.1 – Диаграмма базы данных

Диаграмма базы данных иллюстрирует связи между таблицами и полями, а также отображает различные типы отношений между ними, такие как связи "один-ко-многим", "многие-ко-многим" и "один-к-одному".

Связь "один-ко-многим" указывает на то, что одна запись в одной таблице может иметь множество связанных записей в другой таблице. Например, один исполнитель может иметь много треков.

Связь "многие-ко-многим" означает, что множество записей в одной таблице может быть связано с множеством записей в другой таблице. Например, треков определённого жанра может быть много, и некоторые треки могут быть определены сразу в несколько жанров.

Связь "один-к-одному" указывает на то, что каждая запись в одной таблице соответствует только одной записи в другой таблице и наоборот. Например, у каждого пользователя может быть только один тип подписки одновременно.

**2.2 Описание информационных объектов**

Для реализации базы данных было разработано 18 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: Role, Role\_person, Person, Genre, Genre\_audio, Audio, Audio\_author, Playlist, Playlist\_audio, Playlist\_author, Nravlik, Nravlik\_album, Follows, Json\_buffer, Save\_sudio, Album, Album\_audio, Album\_author. Ниже будет описание про каждую более подробно в виде таблицы.

|  |  |  |
| --- | --- | --- |
| Название таблицы | Название столбцов | Тип хранимых данных |
| audio | id  name  text  image  upload\_date | integer (PK) (NN)  character varying (32) (NN)  character varying (1000)  text  timestamp without time zone(NN) |
| person | id  username  password  email  registration\_date | integer (PK) (NN)  character varying (32) (NN) (UNIQ)  character varying (60) (NN)  character varying (256) (NN) (UNIQ)  timestamp without time zone(NN) |
| playlist | id  name  description  creation\_date  image | integer (PK) (NN)  character varying (128) (NN)  character varying (5000)  date (NN)  text |
| album | id  name  creation\_date  description  image | integer (PK) (NN)  character varying (64) (NN)  date (NN)  character varying (1000)  text |
| role | id  name | integer (PK) (NN)  character varying (32) (NN) (UNIQ) |
| json\_buffer | id  name  text  image  genre\_name  username  upload\_date | numeric (PK) (NN)  character varying (32) (NN)  character varying (1000)  text  character varying (32) (NN)  character varying (32) (NN)  timestamp without time zone(NN) |
| genre | id  genre\_name | integer (PK) (NN)  character varying (32) (NN) (UNIQ) |
| album\_audio | audio\_id  album\_id | integer (FK) (NN)  integer (FK) (NN) |
| album\_author | author\_id  album\_id |
| author\_audio | author\_id  audio\_id |
| follows | person\_id  author\_id |
| genre\_audio | genre\_id  audio\_id |
| nravlik | person\_id  audio\_id |
| nravlik\_albums | person\_id  album\_id |
| playlist\_audio | playlist\_id  audio\_id |
| playlist\_author | playlist\_id  author\_id |
| role\_person | role\_id  person\_id |
| save\_audio | audio\_id  person\_id |

**2.3 Вывод**

При разработке проектной архитектуры имеет большое значение определение структуры и функциональности приложения. Общая структура управления приложением позволяет определить, какие компоненты необходимы для реализации приложения и как они должны взаимодействовать друг с другом.

Важной частью проектной архитектуры является описание информационных объектов, поскольку это позволяет понять, какие данные будут использоваться в приложении, а также как они будут храниться и обрабатываться. В данном проекте были определены объекты, такие как пользователи, роли, жанры, артисты, треки, рейтинги, плейлисты и библиотека пользователя.

В целом, разработка проектной архитектуры является важным этапом в процессе разработки приложения, поскольку она позволяет определить необходимые компоненты и информационные объекты, которые способствуют созданию функционального и эффективного приложения.

3 Разработка модели базы данных

## 3.1 Создание необходимых объектов

Для музыкального сервиса необходимо создать несколько таблиц, которые будут хранить данные о пользователях, жанрах, аудиофайлах, альбомах и плейлистах пользователей.

Для музыкального сервиса необходимо создать следующие таблицы:

AUDIO - таблица, содержащая информацию о музыкальных треках. В этой таблице будут храниться данные о музыкальных треках, такие как название, дата выпуска, ссылка на изображение обложки и т.д.

PERSON - таблица, содержащая данные о пользователях. В этой таблице будут храниться информация о пользователе, такая как псевдоним на сайте, пароль, электронная почта, дата регистрации и т.д.

GENRE - таблица, содержащая жанры музыки. В этой таблице будут музыкальные жанры, используемые при добавлении музыки на сервис.

PLAYLIST - таблица, содержащая информацию о плейлистах пользователей. В этой таблице будут храниться данные о плейлистах, такие как название и пользователь, создавший плейлист, дата создания плейлиста и путь к изображению.

ALBUM - таблица, содержащая информацию об альбомах исполнителей. В этой таблице будут храниться данные об альбомах, такие как название и авторы альбома, дата публикации и ссылка на обложку.

ROLE - таблица, содержащая список ролей, доступных пользователю на сайте. Например, роль слушателя или исполнителя. Таблица содержит названия ролей.

ALBUM\_AUDIO - таблица, связывающая между собой таблицы ALBUM и AUDIO. Хранит информацию о треках, содержащихся в альбомах.

ALBUM\_AUTHOR - таблица, связывающая между собой таблицы ALBUM и PERSON. Хранит информацию об авторах альбомов.

AUTHOR\_AUDIO - таблица, связывающая между собой таблицы PERSON и AUDIO. Содержит в себе информацию об авторах каждого трека базы данных.

GENRE\_AUDIO - таблица, связывающая между собой таблицы GENRE и AUDIO. Содержит список жанров относящихся с каждого треку на сервисе.

FOLLOWS - таблица, связывающая между собой таблицу PERSON. Содержит в себе информацию о подписках на каждого пользователя на сервисе.

NRAVLIK - таблица, связывающая между собой таблицы PERSON и AUDIO. Содержит информацию об отметках “нравится” относящихся к каждому треку.

JSONBUFFER – таблица служащая буфером во время чтения данных о треках из файлов формата JSON. Хранит данные до моменты попытки следующего чтения из файла.

NRAVLIK\_ALBUMS - таблица, связывающая между собой таблицы PERSON и ALBUM. Содержит информацию об отметках “нравится” относящихся к каждому альбому.

PLAYLIST\_AUDIO - таблица, связывающая между собой таблицы PLAYLIST и AUDIO. Хранит информацию о треках, содержащихся в пользовательских плейлистах.

PLAYLIST\_AUTHOR - таблица, связывающая между собой таблицы PLAYLIST и PERSON. Хранит информацию об авторах альбомов.

SAVE\_AUDIO - таблица, связывающая между собой таблицы PERSON и AUDIO. Содержит информацию о треках, сохранённых пользователем локально на устройство.

ROLE\_PERSON - таблица, связывающая между собой таблицы ROLE и PERSON. Содержит информацию о роли, принадлежащей каждому пользователю на сервисе.

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Например, можно создать индекс на столбце NAME в таблице PLAYLIST, чтобы быстро находить информацию о плейлисте по его названию.

Всё взаимодействие пользователя с базой данных необходимо организовать через хранимые процедуры и функции. К примеру добавление и удаление музыки, создание плейлистов, оформление платной подписки и весь прочий функционал.

### 3.1.2 Индексы базы данных

Индекс в базе данных [4] - это объект, который используется для повышения скорости поиска данных. Когда таблица содержит большое количество строк, последовательный поиск данных может занимать значительное время. Индекс создается на основе значений одного или нескольких столбцов таблицы и содержит указатели на соответствующие строки, что позволяет быстро находить строки, удовлетворяющие заданным критериям поиска. Использование индексов ускоряет работу с базой данных, так как они имеют оптимизированную структуру для поиска, например, сбалансированное дерево. Это позволяет эффективно выполнять операции поиска, сортировки и связывания данных в таблице, сокращая время доступа к нужным записям и повышая производительность системы.

Для того, чтобы быстро выполнять поиск в больших таблицах, были созданы пять индексов, располагающиеся на таблицах: audio, person, playlist, album. Сами индексы представлены на листинге 3.2.

|  |
| --- |
| CREATE INDEX IF NOT EXISTS audio\_name\_hash\_idw  ON audio USING hash (name COLLATE pg\_catalog."default")  TABLESPACE musicservicedefault;    CREATE INDEX IF NOT EXISTS id\_btree\_idx  ON playlist USING btree (id ASC NULLS LAST)  TABLESPACE musicservicedefault;  CREATE INDEX IF NOT EXISTS playlist\_name\_hash\_idx  ON playlist USING hash(name COLLATE pg\_catalog."default")  TABLESPACE musicservicedefault;    CREATE INDEX IF NOT EXISTS id\_btree\_index  ON album USING btree(id ASC NULLS LAST)  TABLESPACE musicservicedefault;    CREATE INDEX IF NOT EXISTS name\_hash\_index  ON person USING hash (username COLLATE pg\_catalog."default")  TABLESPACE musicservicedefault;  CREATE INDEX IF NOT EXISTS person\_id\_btree\_index  ON person USING btree (id ASC NULLS LAST)  TABLESPACE musicservicedefault; |

Листинг 3.1 – Индексы базы данных

Индекс AUDIO\_NAME\_HASH\_IDW на таблице AUDIO был создан для ускорения поиска трека по его имени (NAME).

Индекс ID\_BTREE\_IDX на таблице ALBUM может использоваться для быстрого поиска альбомов по их ID.

Индекс PLAYLIST\_NAME\_HASH\_IDX на таблице PLAYLIST был создан для ускорения поиска плейлистов по их именам (NAME).

Индекс NAME\_HASH\_INDEX на таблице PERSON был создан для ускорения поиска пользователей по их именам (NAME).

Индекс PERSON\_ID\_BTREE\_INDEX на таблице PERSON был создан для ускорения поиска пользователей по их ID.

Использование индексов значительно ускоряет операции поиска, сортировки и фильтрации данных в базе данных, особенно при обработке больших объемов данных. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах. Поэтому важно подобрать оптимальное количество и типы индексов для достижения наилучшей производительности базы данных. Нужно учитывать баланс между скоростью выполнения запросов и накладными расходами на обновление индексов. Тщательное планирование и анализ использования индексов помогут создать эффективную структуру базы данных, которая обеспечит быстрый доступ к данным без излишних накладных расходов.

## 3.2 Описание используемой технологии

В данной базе данных используется репликация для обеспечения надежности и доступности данных. Репликация - это процесс создания и поддержания точных копий данных, которые распределены на различных узлах или серверах. Это позволяет достичь повышенной отказоустойчивости, улучшить производительность и обеспечить более быстрый доступ к данным.

В случае репликации данных, изменения, внесенные в одной копии данных, автоматически реплицируются на другие узлы. Например, в базе данных музыкальной платформы, таблица PERSON может быть реплицирована на несколько серверов, чтобы обеспечить доступность профилей пользователей. Таким образом, если один сервер недоступен, данные о пользователях все равно будут доступны на других серверах.

Для реализации данной технологии в PostgreSQL 15 необходимо создать специального пользователя-репликатора [4] на основном сервере и на всех репликах, куда мы хотим отправлять данные. Пользователь должен иметь специальное разрешение REPLICATION чтобы иметь возможность реализовывать репликацию.

Код создание пользователя приведён в листинге 3.3

|  |
| --- |
| CREATE USER replica\_user WITH PASSWORD 'password123';  ALTER USER replica\_user WITH REPLICATION LOGIN; |

Листинг 3.2 – создание пользователя-репликатора

После создание пользователя репликатора на серверах, которые необходимо связать, надо изменить файлы параметров PostgreSQL.

В первую очередь необходимо в файле postresql.conf поменять значение параметра wal\_level, установив его на logical. Данное действие активирует логическу репликацию. Данное действие необходимо проделать на основном сервере и на репликах.

Далее необходимо создать публикацию на основном сервере, в ней мы будем определять данные, которые в дальнейшем будут реплицироваться. В случае данного проекта реплицироваться будут данные всех из всех таблиц.

Код создание публикации приведён в листинге 3.4

|  |
| --- |
| CREATE PUBLICATION my\_publication FOR ALL TABLES; |

Листинг 3.3 – создание публикации на основном сервере

После создание публикации необходимо к ней подключиться со всех реплик, на которые мы будем отправлять данные. Для этого необходимо создать подписку на репликах с указанием необходимых параметров подключения к основному серверу.

Код создания подписки приведён в листинге 3.5

|  |
| --- |
| CREATE SUBSCRIPTION replica\_subs  CONNECTION 'host=172.30.208.1 port=5432 dbname=MusicDB user=replica\_user password=password123'  PUBLICATION my\_publication; |

Листинг 3.5 – создание подписки на публикацию

Репликация данных в базе данных музыкальной платформы способствует обеспечению надежности и доступности мультимедийных данных. При этом следует учитывать, что репликация требует дополнительных ресурсов и может привести к небольшой задержке при обновлении данных. Оптимальное планирование и настройка репликации позволяет достичь баланса между надежностью и производительностью системы.

## 3.3 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для музыкального сервиса, а также была описана использованная технология репликации для обеспечения надежности и доступности данных.

Репликация позволяет создать копии базы данных на разных серверах, что обеспечивает резервное хранение данных и возможность их использования в случае сбоев или недоступности основного сервера. Таким образом, использование данной технологии повышает надежность и устойчивость работы музыкального сервиса.

4 Установка, настройка и использование PosgtreSQL 15

## 4.1 Установка PostgreSQL

PostgreSQL - это мощная реляционная база данных с открытым исходным кодом, предлагающая широкий спектр функций, включая многопоточность, транзакционность, контроль целостности данных, масштабируемость и многое другое. Она находит применение в различных коммерческих и научных проектах, а также веб- и мобильных приложениях.

После успешной установки PostgreSQL на сервер [5] была произведена конфигурация для обеспечения оптимальной работы с базой данных. В рамках конфигурации была создана база данных с названием MusicDB, которая будет использоваться для хранения данных музыкального приложения. Эта база данных предоставляет среду, в которой будут храниться и организовываться информационные объекты, такие как пользователи, треки, жанры, альбомы и другие связанные данные, необходимые для функционирования сервиса.

## 4.2 Создание таблиц

В данном разделе мы сосредоточимся на создании таблиц для нашей базы данных. Однако, перед тем как приступить к созданию таблиц, необходимо создать табличное пространство.

Табличное пространство представляет собой механизм, который связывает объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на различные физические устройства или диски, что может значительно улучшить производительность работы с базой данных.

Скрипт для создания табличного пространства представлен в листинге 4.1.

|  |
| --- |
| CREATE TABLESPACE musicservicedefault  OWNER postgres  LOCATION 'C:\Program Files\PostgreSQL\15\data\tbs'; |

Листинг 4.1 – Cкрипт для создания табличного пространств

Каждая таблица будет иметь свои поля (столбцы) и ограничения (constraints), которые определяют правила для хранения и изменения данных. Ограничение FOREIGN KEY, например, устанавливает связь между двумя таблицами, а ограничение PRIMARY KEY определяет уникальный идентификатор для каждой записи в таблице.

Кроме того, база данных будет содержать связи между таблицами. Одним из основных типов связи является "один ко многим" (one-to-many), которая описывает отношение одной записи в одной таблице к нескольким записям в другой таблице. Например, каждый пользователь может иметь множество треков, добавленных в его библиотеку. Для этого мы добавим в таблицу "NRAVLICK" внешний ключ (FOREIGN KEY), указывающий на идентификатор пользователя.

Другим типом связи является "многие ко многим" (many-to-many), который определяет отношение между множеством записей в одной таблице и множеством записей в другой таблице. Например, каждый пользователь может иметь множество плейлистов, а каждый плейлист может содержать множество треков. Для этого мы создадим таблицу "PLAYLIST\_AUDIO", которая будет содержать внешние ключи на таблицы "PLAYLIST" и "AUDIO".

Скрипт создания таблицы AUDIO будет представлен в листинге 4.2.

|  |
| --- |
| CREATE TABLE IF NOT EXISTS public.audio  (  id integer NOT NULL DEFAULT 'nextval('audio\_id\_seq'::regclass)',  name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  text character varying(1000) COLLATE pg\_catalog."default",  image text COLLATE pg\_catalog."default",  upload\_date timestamp without time zone NOT NULL,  CONSTRAINT audio\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault; |

Листинг 4.2 – Cкрипт создание таблицы User

Грамотно построенная структура базы данных с правильными связями между таблиц позволяет облегчить доступ к данным, а также хранить их в уобном, что несомненно покажет только положительный результат при большой нагрузке на базу данных. Таким образом было описано создание табличного пространства и таблиц базы данных и приведён пример скрипта создания таблицы и табличного пространства.

## Создание ролей для разграничения

В данном разделе производится создание ролей для управления доступом к базе данных. Создание ролей позволяет установить ограничения на доступ к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Это важный шаг для обеспечения безопасности данных и контроля доступа в базе данных.

Будут созданы три роли для разграничения доступа к базе данных: AUTHOR, ADMINISTRATOR, LISTENER. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

Роль ADMINISTRATOR имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выдачу привилегии роли ADMINISTARTOR можно увидеть на листинге 4.3.

|  |
| --- |
| GRANT ALL PRIVILEGES ON DATABASE "MusicDB" TO administrator; |

Листинг 4.3 – Привилегии, выданные роли ADMINISTARTOR

Роль AUTHOR имеет возможность добавлять и изменять треки и альбомы, создавать плейлисты, имеет доступ к процедурам и функциям, реализующим добавление, изменение и удаление данных необходимым пользователя. Выданные привилегии роли AUTHOR можно увидеть в листинге 4.4.

|  |
| --- |
| GRANT SELECT, INSERT, DELETE, UPDATE ON audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON author\_audio to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON genre\_audio to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON nravlik to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON nravlik\_albums to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON follows to Author;  GRANT SELECT ON genre to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON album to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON album\_audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON album\_author to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON save\_audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON person to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_author to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON role\_person to Author;  GRANT SELECT ON role to Author;  GRANT SELECT, USAGE ON person\_id\_seq to Author;  GRANT SELECT ON genre\_id\_seq to Author;  GRANT SELECT, USAGE ON audio\_id\_seq to Author;  GRANT SELECT, USAGE ON playlist\_id\_seq to Author;  GRANT SELECT ON role\_id\_seq to Author;  GRANT SELECT, USAGE ON album\_id\_seq to Author; |

Листинг 4.4 – Привилегии, выданные роли AUTHOR

Роль LISTENER может добавлять свои треки и редактировать их, создавать плейлисты и добавлять в них треки или добавлять треки в свою библиотеку. Кроме того, LISTENER имеет возможность изменять свой профиль и пароль. Роль также может оценивать треки и удалять их из своих плейлистов или медиатеки. Пользователь может просматривать профили других пользователей, искать треки по названию или имени пользователя и просматривать информацию о треках, такую как автор, название, жанр и т.д. Выданные привилегии роли LISTENER можно увидеть на листинге 4.5.

|  |
| --- |
| GRANT SELECT, INSERT, UPDATE, DELETE ON nravlik\_albums to Listener;  GRANT SELECT, INSERT, UPDATE, DELETE ON follows to Listener;  GRANT SELECT ON album to Listener;  GRANT SELECT ON album\_audio to Listener;  GRANT SELECT ON album\_author to Listener;  GRANT SELECT ON audio to Listener;  GRANT SELECT ON author\_audio to Listener;  GRANT SELECT ON genre\_audio to Listener;  GRANT SELECT ON genre to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON save\_audio to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON nravlik to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON person to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_audio to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_author to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON role\_person to Listener;  GRANT SELECT ON role to Listener;  GRANT SELECT, USAGE ON person\_id\_seq to Listener;  GRANT SELECT, USAGE ON genre\_id\_seq to Listener;  GRANT SELECT, USAGE ON audio\_id\_seq to Listener;  GRANT SELECT, USAGE ON playlist\_id\_seq to Listener;  GRANT SELECT, USAGE ON role\_id\_seq to Listener; |

Листинг 4.5 – Привилегии, выданные роли LISTENER

Таким образом, были созданы три роли для ограничения доступа к базе данных: AUTHOR, ADMINISTRATOR, LISTENER. Каждая роль имеет определенный набор привилегий, который позволяет пользователю выполнять определенные функции в базе данных. Роль ADMINISTRATOR имеет наибольшие привилегии и может выполнять любые операции в базе данных, в то время как роль LISTENER имеет ограниченный набор привилегий, который позволяет пользователю только просматривать информацию, слушать музыку и создавать свои плейлисты. Роль AUTHOR сохраняет весь функционал роли LISTENER и расширяет его возможность добавлять, изменять и удалять музыку и альбомы в базе данных.

### 4.4 Выборка данных из таблиц

Для вывода данных из таблиц были написаны следующие функции [3]: get\_all\_audios\_by\_author\_name, get\_audio\_info\_by\_audio\_id, get\_all\_audios\_by\_genre, get\_all\_audios\_from\_playlist\_by\_playlist\_id, get\_amount\_of\_nravlicks\_by\_audio\_id, get\_audio\_by\_name, get\_info\_by\_username, get\_playlists\_by\_name.

Основная их задача – выборка данных из всех основных таблиц базы данных. Ниже будут описание каждой функции.

get\_all\_audios\_by\_author\_name, get\_all\_audios\_by\_genre – функции, предназначенные для поиска всех песен по жанру или исполнителю.

get\_all\_audios\_from\_playlist\_by\_playlist\_id – функция, использующаяся для получения списка всех треков в плейлисте.

get\_amount\_of\_nravlicks\_by\_audio\_id – функция, отображающая общее количество оценок, поставленных пользователями треку.

get\_audio\_by\_name - функции для поиска треков по названию.

get\_audio\_info\_by\_audio\_id - функция для получения подробной информации о треке.

get\_info\_by\_username - функция для получения информации о пользователе по его псевдониму.

get\_playlists\_by\_name – функция для поиска плейлистов по названию.

На листинге 4.6 будет представлена реализация функции get\_all\_audios\_by\_genre, которая предназначена для выборки всех песен определённого жанра.

|  |
| --- |
| CREATE OR REPLACE FUNCTION public.get\_all\_audios\_by\_genre(  \_genre\_name text)  RETURNS TABLE(audio\_id integer, audio\_name character varying, text character varying, audio\_upload\_date timestamp without time zone, authors text)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT a.id, a.name, a.text, a.upload\_date, string\_agg(p.username, ', ')  FROM genre\_audio ga  JOIN genre g ON ga.genre\_id = g.id  JOIN audio a ON ga.audio\_id = a.id  JOIN author\_audio aa on a.id = aa.audio\_id  JOIN person p on p.id = aa.author\_id  WHERE (g.genre\_name = \_genre\_name)  GROUP BY a.id;  END;  $BODY$; |

Листинг 4.6 – Функция get\_all\_audios\_by\_genre

Все остальные функции будут аналогичны и также предназначены для выборки данных из различных таблиц базы данных.

### 4.4.1 Заполнение таблиц 100 000 строк

Для заполнения таблицы PLAYLIST была разработана процедура INSERTCYCLEPROCEDURE, которая вставляет заданное количество строк в таблицу. Код процедуры представлен на листинге 4.7.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.insertcycleprocedure(  IN cycle\_count numeric)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  FOR counter IN 1..cycle\_count  LOOP  INSERT INTO playlist(id, name, description, creation\_date)  VALUES (counter, 'stringType', 'Description', '2023-04-04');  END LOOP;  INSERT INTO playlist  (id, name, description, creation\_date)  VALUES  (658324, 'Jazz', 'Jazz playlis', '2023-05-04'),  (836853, 'Hip-Hop', 'Hip-Hop playlist', '2023-05-05'),  (386392, 'Pop', 'Pop playlist', '2023-05-07'),  (547283, 'Classic', 'Classic playlist', '2023-05-06'),  (174392, 'Dub', 'Dub playlist', '2023-05-08');  END;  $BODY$;  ALTER PROCEDURE public.insertcycleprocedure(numeric)  OWNER TO administrator; |

Листинг 4.7 – Процедура заполнения таблицы PLAYLIST

Процедура INSERTCYCLEPROCEDURE была создана для заполнения таблицы PLAYLIST 100000 строк. В теле функции используется цикл FOR, который проходит по значениям от 1 до заданного значения и для каждого значения выполняет вставку новой строки в таблицу PLAYLIST с заданными аргументы и ID соответствующим номеру итерации. Выполняется процедура обычным способом, непобходимо вызвать её в SQL-скрипте с помощью оператора CALL

После проведённых опытов с поиском данных в заполненной таблице в неё был добавлен дополнительный индекс, ускоряющий работу с ней в два раза при наших значениях. Также стоит упомянуть, что была разработана процедура, полностью очищающая заполненную таблицу.

### 4.4.2 Добавление данных в таблицы

Были разработаны следующие процедуры [3] для добавления новых строк в основные таблицы базы данных: NEW\_ALBUM, NEW\_AUDIO, NEW\_FOLLOW, NEW\_GENRE, NEW\_NRAVLIK, NEW\_NRAVLIK\_ALBUM, NEW\_PERSON, NEW\_PLAYLIST, SAVE\_AUDIO. Ниже будет описание каждой процедуры.

NEW\_PERSON - процедура, которая добавляет нового пользователя в таблицу PERSON. Принимает значения для полей \_lname, \_password, \_email.

NEW\_AUDIO - процедура, которая добавляет новый трек в таблицу AUDIO. Принимает значения для полей \_name, \_text, \_image, user\_names, genres.

NEW\_PLAYLIST - процедура, которая добавляет новый плейлист в таблицу PLAYLIST. Принимает значения для полей playlist\_name, playlist\_description, \_image, audios, authors.

NEW\_GENRE - процедура, которая добавляет новый жанр в таблицу GENRE. Принимает значение для поля \_name.

NEW\_NRAVLIK - процедура, которая добавляет новую отметку “нравится” в таблицу NRAVLIK. Принимает значения для полей \_audio\_id, \_person\_id.

NEW\_NRAVLIK\_ALBUMS - процедура, которая добавляет новую отметку “нравится” в таблицу NRAVLIK\_ALBUMS. Принимает значения для полей \_album\_id, \_person\_id.

NEW\_FOLLOW - процедура, которая добавляет новую подписку в таблицу FOLLOWS. Принимает значения для полей \_person\_id, \_author\_id.

NEW\_ALBUM - процедура, которая добавляет новый альбом в таблицу ALBUM. Принимает значения для полей album\_name, album\_description, \_image, audios, authors.

SAVE\_AUDIO - процедура, которая сохраняет локальную копию трека. Принимает значения для полей pers\_id, aud\_id.

На листинге 4.8 будет функция NEW\_PERSON, которая добавляет пользователя в таблицу PERSON.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.new\_person(  IN \_lname text,  IN \_password text,  IN \_email text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  res person;  BEGIN  INSERT INTO person(username, password, email, registration\_date)  VALUES(\_lname, md5(\_password), \_email, now())  RETURNING \* INTO res;  INSERT INTO role\_person (person\_id, role\_id)  VALUES (res.id, (SELECT p.id FROM role p WHERE p.name = 'Default'));  END;  $BODY$; |

Листинг 4.8 – Процедура заполнения таблицы NEW\_PERSON

Все остальные процедуры будут аналогичны, также предназначены для добавления новых строк в основные таблицы базы данных.

### 4.4.3 Удаление данных из таблиц

Для удаления данных из базы данных были созданы процедуры: DELETE\_ALBUM, DELETE\_AUDIO, DELETE\_FOLLOW, DELETE\_GENRE, DELETE\_NRAVLIK, DELETE\_NRAVLIK\_ALBUM, DELETE\_PERSON, DELETE\_PLAYLIST, DELETE\_SAVED\_AUDIO. Ниже будет описание каждой процедуры.

DELETE\_PERSON - процедура, которая удаляет пользователя из таблицы PERSON. Принимает значения для полей \_user\_name.

DELETE\_AUDIO - процедура, которая удаляет трек из таблицы AUDIO. Принимает значения для полей \_audio\_id.

DELETE\_PLAYLIST - процедура, которая удаляет плейлист из таблицы PLAYLIST. Принимает значения для полей \_id.

DELETE\_GENRE - процедура, которая удаляет жанр из таблицы GENRE. Принимает значение для поля \_name.

DELETE\_NRAVLIK - процедура, которая удаляет запись о лайке из таблицы NRAVLIK. Принимает значения для полей \_audio\_id, \_person\_id.

DELETE\_NRAVLIK\_ALBUMS - процедура, которая удаляет запись о лайке из таблицы NRAVLIK\_ALBUMS. Принимает значения для полей \_album\_id, \_person\_id.

DELETE\_FOLLOW - процедура, которая удаляет запись о подписке из таблицы FOLLOWS. Принимает значения для полей \_person\_id, \_author\_id.

DELETE\_ALBUM - процедура, которая удаляет альбом из таблицы ALBUM. Принимает значения для полей \_id.

DELETE\_SAVED\_AUDIO - процедура, которая удаляет локальную копию трека. Принимает значения для полей pers\_id, aud\_id.

На листинге 4.9 будет процедура DELETE\_PERSON.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.delete\_person(  IN user\_name text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE user\_id int;  BEGIN  SELECT person.id INTO STRICT user\_id  FROM person  WHERE username = user\_name;  DELETE FROM nravlik WHERE person\_id = user\_id;    DELETE FROM audio WHERE id IN  (SELECT audio\_id FROM author\_audio WHERE author\_id = user\_id);  DELETE FROM author\_audio WHERE author\_id = user\_id;    DELETE FROM playlist WHERE id IN  (SELECT playlist\_id FROM playlist\_author WHERE author\_id = user\_id);  DELETE FROM playlist\_audio WHERE playlist\_id IN  (SELECT playlist\_id FROM playlist\_author WHERE author\_id = user\_id);    DELETE FROM playlist\_author WHERE author\_id = user\_id;    DELETE FROM role\_person WHERE person\_id = user\_id;  DELETE FROM save\_audio WHERE person\_id = user\_id;  DELETE FROM person WHERE id = user\_id;  END;  $BODY$; |

Листинг 4.9 – Процедура для удаления PERSON

Все остальные процедуры будут аналогичны, также предназначены для удаления соответствующих данных из основных таблиц базы данных.

### 4.4.4 Изменение данных в таблицах

Для изменения данных из базы данных были созданы процедуры: CHANGE\_ALBUM, CHANGE\_AUDIO, CHANGE\_GENRE, CHANGE\_PERSON, CHANGE\_PLAYLIST. Ниже будет описание каждой процедуры.

CHANGE\_PERSON - процедура, которая изменяет пользователя в таблице PERSON. Принимает значения для полей \_lname, \_password, \_email, \_role.

CHANGE\_AUDIO - процедура, которая удаляет трек из таблицы AUDIO. Принимает значения для полей \_id, \_name, \_text, \_image, genres, user\_names, \_upload\_date.

CHANGE\_PLAYLIST - процедура, которая удаляет плейлист из таблицы PLAYLIST. Принимает значения для полей \_id, \_name, \_description, \_image, \_audios.

CHANGE\_GENRE - процедура, которая изменяет жанр в таблице GENRE. Принимает значение для поля \_id, \_name.

CHANGE\_ALBUM - процедура, которая изменят альбом в таблице ALBUM. Принимает значения для полей \_id, \_name, \_description, \_image, \_audios, \_user\_names, \_creation\_date.

На листинге 4.10 будет процедура CHANGE\_PERSON.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE UPDATE\_USER(IN\_USER\_ID INTEGER,  IN\_USER\_NAM VARCHAR(255),  IN\_USER\_DATE\_OF\_BIRTH DATE,  IN\_USER\_IMG BYTEA)  LANGUAGE PLPGSQL AS $$  BEGIN  IF IN\_USER\_IMG IS NOT NULL THEN  UPDATE USERS  SET USER\_NAME = IN\_USER\_NAME,  USER\_DATE\_OF\_BIRTH = IN\_USER\_DATE\_OF\_BIRTH,  USER\_IMG = IN\_USER\_IMG  WHERE USER\_ID = IN\_USER\_ID;  ELSE  UPDATE USERS  SET USER\_NAME = IN\_USER\_NAME,  USER\_DATE\_OF\_BIRTH = IN\_USER\_DATE\_OF\_BIRTH  WHERE USER\_ID = IN\_USER\_ID;  END IF;  END;  $$; |

Листинг 4.10 – Процедура для обновления пользователя

Все остальные процедуры будут аналогичны, также предназначены для изменения соответствующих данных в основных таблицах базы данных.

## 4.5 Описание процедур экспорта и импорта

База данных имеет возможность экспортировать и импортировать данные для таблицы AUDIO и смежных для неё таблиц AUTHOR\_ AUDIO и GENRE\_AUDIO в формат JSON. Это может быть полезно в случае необходимости переноса данных на другой сервер или резервного копирования данных.

Для экспорта данных в формате JSON была создана процедура EXPORT\_TO\_JSON\_TRACKS. Процедура представлена на листинге 4.11.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.export\_to\_json\_tracks()  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  COPY (  SELECT jsonb\_agg(gt) FROM (  SELECT audio.id, audio.name, audio.text, audio.upload\_date,  audio.image, genre.genre\_name, person.username  FROM audio  INNER JOIN genre\_audio  ON audio.id = genre\_audio.audio\_id  INNER JOIN genre  ON genre\_audio.genre\_id = genre.id  INNER JOIN author\_audio  ON audio.id = author\_audio.audio\_id  INNER JOIN person  ON author\_audio.author\_id = person.id  ORDER BY audio.id  ) gt  ) TO 'C:\Program Files\PostgreSQL\15\data\ee\tracks.json';  END;  $BODY$; |

Листинг 4.11 – процедура EXPORT\_TO\_JSON\_TRACKS

Для импорта данных из файла в формате JSON была создана процедура IMPORT\_FROM\_JSON\_TRACKS которая принимает путь к файлу в качестве входного параметра и импортирует данные из файла в таблицы AUDIO, AUTHOR\_AUDIO и GENRE\_AUDIO.

Процедура представлена на листинге 4.12.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE public.import\_from\_json\_tracks(  IN path\_to\_file text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  json\_string jsonb;  BEGIN  json\_string := pg\_read\_file(path\_to\_file);    DELETE FROM json\_buffer;    INSERT INTO json\_buffer (id, name, text, upload\_date, image, genre\_name, username)  SELECT (obj->>'id')::numeric, obj->>'name', obj->>'text', (obj->>'upload\_date')::timestamp, obj->>'image', obj->>'genre\_name', obj->>'username'  FROM jsonb\_array\_elements(json\_string) AS obj;  INSERT INTO audio (id, name, text, upload\_date, image)  SELECT id, name, text, upload\_date, image  FROM json\_buffer;  INSERT INTO genre\_audio(audio\_id, genre\_id)  SELECT json\_buffer.id, genre.id  FROM json\_buffer  INNER JOIN genre  ON json\_buffer.genre\_name = genre.genre\_name;  INSERT INTO author\_audio(audio\_id, author\_id)  SELECT json\_buffer.id, person.id  FROM json\_buffer  INNER JOIN person  ON json\_buffer.username = person.username;  END;  $BODY$; |

Листинг 4.12 – Функция IMPORT\_FROM\_JSON\_TRACKS

Таким образом, администратор базы данных могут легко экспортировать и импортировать данные в формат JSON, что делает управление базой данных более удобным и эффективным.

## Вывод

В данном разделе были рассмотрены основные этапы установки, настройки и использования PostgreSQL. Мы изучили процедуры создания таблиц, создания ролей для ограничения доступа и создания пакетов процедур для выполнения различных операций с данными. Создали процедуры импорта и экспорта с использованием формата JSON и проверили скорость поиска по заполненной большим количеством строк таблице.

В итоге, можно заключить, что PostgreSQL является мощной и надежной системой управления базами данных, которая может успешно обрабатывать большие объемы данных. Правильная установка, настройка и оптимизация запросов позволят достичь высокой производительности и эффективности работы с базой данных.

5 Тестирование

## 5.1 Тестирование производительности базы данных

Тестирование производительности является неотъемлемым этапом разработки, поскольку позволяет оценить эффективность работы базы данных и ее способность обрабатывать запросы с высокой скоростью. Одним из ключевых аспектов такого тестирования является выбор подходящей таблицы, содержащей большой объем данных, чтобы достоверно симулировать реальные условия использования.

В контексте тестирования производительности базы данных, мы выбрали таблицу PLAYLIST для проведения испытаний. Это решение основано на ее размере и структуре данных, которые представляют собой типичные сценарии использования. Посредством использования этой таблицы мы сможем оценить скорость обработки запросов и время, затрачиваемое на получение результатов.

Для получения выборки данных использовался запрос, который представлен на листинге 5.1.

|  |
| --- |
| EXPLAIN ANALYZE SELECT id FROM playlist WHERE id > 75000; |

Листинг 5.1 – Запрос к таблице PLAYLIST

Запрос к таблице PLAYLIST показывает, что его стоимость выполнения равна 16926.00. Это говорит нам о том, что выполнение запроса требует значительных затрат времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время выполнения запроса составляет 70.416мс. Результаты запроса будут представлены на рисунке 5.2.

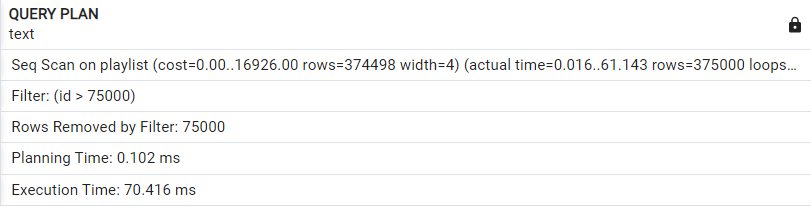


Рисунок 5.2 – Результат выполнения запроса

Для ускорения данного процесса можно создать индекс, использующий алгоритм поиска бинарного дерева на поле ID, так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.3

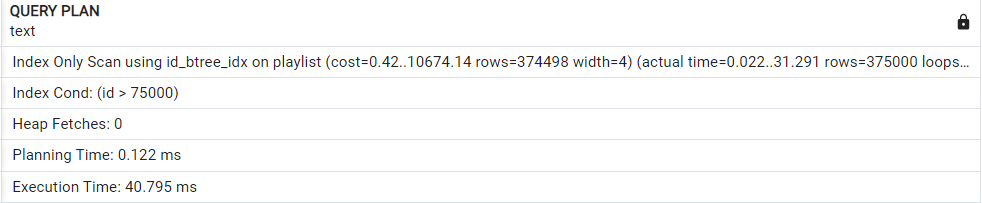


Рисунок 5.3 – Результат выполнения запроса

После создания индекса на поле ID выполнение запроса к таблице PLAYLIST стало более эффективным. Стоимость выполнения запроса сократилась до 10574.14, что является значительным улучшением по сравнению с предыдущим запросом. Время выполнения запроса сократилось до 40.795мс, что почти в два раза быстрее.

Результаты тестирования показали, что создание индекса на поле ID существенно улучшило производительность запроса к таблице PLAYLIST. Стоимость выполнения запроса снизилась с 16926.00 до 10574.14, что свидетельствует о более эффективном сканировании таблицы и применении фильтра. Время выполнения запроса сократилось с 70.416мс до 40.795мс, что представляет собой заметное улучшение скорости работы, особенно учитывая, что в таблице всего 450000 строк. Таким образом, создание индексов на полях, по которым выполняются частые запросы, значительно повышает производительность базы данных.

## 5.2 Вывод

В данном разделе мы исследовали важное понятие тестирования производительности базы данных. Для проведения тестов мы выбрали таблицу PLAYLIST, содержащую значительный объем данных. Проведенные эксперименты подтвердили, что создание индекса на поле ID значительно улучшает производительность запросов к таблице PLAYLIST, сокращая время выполнения и снижая стоимость запросов. Из этого следует вывод, что создание индексов на полях, по которым выполняются частые запросы, способно существенно повысить производительность базы данных, особенно при работе с большими объемами данных.

# Заключение

База данных является ключевым элементом любой современной организации, обеспечивая надежное хранение и управление информацией. В данной работе была поставлена задача разработки базы данных для музыкальной площадки с использованием технологии применения мультимедийных типов данных в СУБД PostgreSQL.

В процессе выполнения работы были использованы различные объекты, включая таблицы, процедуры и функции, чтобы обеспечить структурированное хранение данных и своевременный доступ к ним. Тщательно спроектированная схема базы данных позволяет эффективно хранить информацию о треках, артистах, альбомах, пользовательских предпочтениях и других сущностях музыкальной площадки.

В результате усилий, цель работы была успешно достигнута, и база данных готова к использованию. Для проверки производительности и надежности системы было проведено тестирование базы данных с использованием большого объема данных, и результаты оказались положительными. Система продемонстрировала стабильность и высокую производительность при обработке запросов, обеспечивая своевременный доступ к музыкальным данным.

Для обеспечения удобства использования и управления данными были реализованы процедуры для импорта и экспорта данных в формате JSON. Это позволяет удобно обмениваться данными с другими системами и обеспечивает гибкость в работе с информацией.

Одной из ключевых особенностей разработанной базы данных является технология репликации. Эта технология позволяет дублировать данные на дополнительные сервера, обеспечивая большую безопасность и надежность сервиса. Репликация базы данных помогает обеспечить высокую доступность и отказоустойчивость системы, минимизируя потенциальные проблемы с пропускной способностью и сбоями оборудования.

В целом, результаты работы говорят о том, что база данных полностью соответствует поставленным требованиям и может успешно использоваться в рамках музыкального сервиса. Она обеспечивает надежное хранение, эффективное управление и быстрый доступ к музыкальным данным, что является важным фактором для эффективной работы и развития музыкальной платформы.

# Список литературных источников

1. Spotify [Электронный ресурс] / Режим доступа: <https://open.spotify.com/> – Дата доступа: 12.04.2023.

2. Soundcloud [Электронный ресурс] / Режим доступа: [https://soundcloud.com/](https://soundcloud.com/discover) – Дата доступа: 12.04.2023.

3. PostgreSQL [Электронный ресурс] / Режим доступа: [PostgreSQL: Documentation: 15: Chapter 43. PL/pgSQL — SQL Procedural Language](https://www.postgresql.org/docs/15/plpgsql.html) – Дата доступа: 13.05.2023.

4. OpenAI [Электронный ресурс] / Режим доступа: <https://chat.openai.com/> – Дата доступа: 13.05.2023.

5.  Stackoverflow [Электронный ресурс] / Режим доступа: [postgresql - unable to connect to server for Postgres - Stack Overflow](https://stackoverflow.com/questions/40532399/unable-to-connect-to-server-for-postgres) – Дата доступа: 14.05.2023.

# Приложение А

|  |
| --- |
| --------------------- TABLESPACE ---------------------------  CREATE TABLESPACE musicservicedefault  OWNER postgres  LOCATION 'C:\Program Files\PostgreSQL\15\data\tbs';  ALTER TABLESPACE musicservicedefault  OWNER TO postgres;  /\*---------------------————————————————---------------------  -----------------------| CREATE TABLES |--------------------  ------------------------————————————————-------------------\*/  --------------------- TABLE ALBUM ----------------------  CREATE TABLE IF NOT EXISTS public.album  (  id integer NOT NULL DEFAULT 'nextval('album\_id\_seq'::regclass)',  name character varying(64) COLLATE pg\_catalog."default" NOT NULL,  creation\_date date NOT NULL,  image text COLLATE pg\_catalog."default",  description character varying(1000) COLLATE pg\_catalog."default",  CONSTRAINT album\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault;  --------------------- TABLE AUDIO ----------------------  CREATE TABLE IF NOT EXISTS public.audio  (  id integer NOT NULL DEFAULT 'nextval('audio\_id\_seq'::regclass)',  name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  text character varying(1000) COLLATE pg\_catalog."default",  image text COLLATE pg\_catalog."default",  upload\_date timestamp without time zone NOT NULL,  CONSTRAINT audio\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault;  --------------------- TABLE GENRE ----------------------  CREATE TABLE IF NOT EXISTS public.genre  (  id integer NOT NULL DEFAULT 'nextval('genre\_id\_seq'::regclass)',  genre\_name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  CONSTRAINT genre\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT genre\_name\_key UNIQUE (genre\_name)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault;  --------------------- TABLE PLAYLIST ----------------------  CREATE TABLE IF NOT EXISTS public.playlist  (  id integer NOT NULL DEFAULT 'nextval('playlist\_id\_seq'::regclass)',  name character varying(128) COLLATE pg\_catalog."default" NOT NULL,  description character varying(5000) COLLATE pg\_catalog."default",  creation\_date date NOT NULL,  image text COLLATE pg\_catalog."default",  CONSTRAINT playlist\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault;  --------------------- TABLE ROLE ----------------------  CREATE TABLE IF NOT EXISTS public.role  (  id integer NOT NULL DEFAULT 'nextval('role\_id\_seq'::regclass)',  name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  CONSTRAINT role\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT role\_name\_key UNIQUE (name)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault;  --------------------- TABLE PERSON----------------------  CREATE TABLE IF NOT EXISTS public.person  (  id integer NOT NULL DEFAULT 'nextval('person\_id\_seq'::regclass)',  username character varying(32) COLLATE pg\_catalog."default" NOT NULL,  password character varying(60) COLLATE pg\_catalog."default" NOT NULL,  email character varying(256) COLLATE pg\_catalog."default" NOT NULL,  registration\_date timestamp without time zone NOT NULL,  CONSTRAINT person\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT person\_email\_key UNIQUE (email)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT person\_username\_key UNIQUE (username)  USING INDEX TABLESPACE musicservicedefault  )TABLESPACE musicservicedefault;  --------------------- TABLE ALBUM\_AUDIO ----------------------  CREATE TABLE IF NOT EXISTS public.album\_audio  (  audio\_id integer NOT NULL,  album\_id integer NOT NULL,  CONSTRAINT album\_audio\_pkey PRIMARY KEY (audio\_id, album\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT album\_audio\_album\_id\_fkey FOREIGN KEY (album\_id)  REFERENCES public.album (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  NOT VALID,  CONSTRAINT album\_audio\_audio\_id\_fkey FOREIGN KEY (audio\_id)  REFERENCES public.audio (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  NOT VALID  ) TABLESPACE musicservicedefault;  --------------------- TABLE ALBUM\_AUTHOR ----------------------  CREATE TABLE IF NOT EXISTS public.album\_author  (  author\_id integer NOT NULL,  album\_id integer NOT NULL,  CONSTRAINT album\_auth\_pkey PRIMARY KEY (author\_id, album\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT album\_author\_album\_id\_fkey FOREIGN KEY (album\_id)  REFERENCES public.album (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  NOT VALID,  CONSTRAINT album\_author\_author\_id\_fkey FOREIGN KEY (author\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  NOT VALID  ) TABLESPACE musicservicedefault;  --------------------- TABLE AUTHOR\_AUDIO ----------------------  CREATE TABLE IF NOT EXISTS public.author\_audio  (  author\_id integer NOT NULL,  audio\_id integer NOT NULL,  CONSTRAINT author\_audio\_pkey PRIMARY KEY (author\_id, audio\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT author\_audio\_audio\_id\_fkey FOREIGN KEY (audio\_id)  REFERENCES public.audio (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT author\_audio\_author\_id\_fkey FOREIGN KEY (author\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault;  --------------------- TABLE FOLLOWS ----------------------  CREATE TABLE IF NOT EXISTS public.follows  (  author\_id integer NOT NULL,  person\_id integer NOT NULL,  CONSTRAINT follows\_pkey PRIMARY KEY (author\_id, person\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT follows\_author\_id\_fkey FOREIGN KEY (author\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  NOT VALID,  CONSTRAINT follows\_person\_id\_fkey FOREIGN KEY (person\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE NO ACTION  NOT VALID  ) TABLESPACE musicservicedefault;  --------------------- TABLE AUTHOR\_AUDIO ----------------------  CREATE TABLE IF NOT EXISTS public.author\_audio  (  author\_id integer NOT NULL,  audio\_id integer NOT NULL,  CONSTRAINT author\_audio\_pkey PRIMARY KEY (author\_id, audio\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT author\_audio\_audio\_id\_fkey FOREIGN KEY (audio\_id)  REFERENCES public.audio (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT author\_audio\_author\_id\_fkey FOREIGN KEY (author\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault;  --------------------- TABLE GENRE\_AUDIO ----------------------  CREATE TABLE IF NOT EXISTS public.genre\_audio  (  genre\_id integer NOT NULL,  audio\_id integer NOT NULL,  CONSTRAINT genre\_audio\_pkey PRIMARY KEY (genre\_id, audio\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT genre\_audio\_audio\_id\_fkey FOREIGN KEY (audio\_id)  REFERENCES public.audio (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT genre\_audio\_genre\_id\_fkey FOREIGN KEY (genre\_id)  REFERENCES public.genre (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault;  --------------------- TABLE JSON\_BUFFER ----------------------  CREATE TABLE IF NOT EXISTS public.json\_buffer  (  id numeric NOT NULL,  name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  text character varying(1000) COLLATE pg\_catalog."default",  image text COLLATE pg\_catalog."default",  genre\_name character varying(32) COLLATE pg\_catalog."default" NOT NULL,  username character varying(32) COLLATE pg\_catalog."default" NOT NULL,  upload\_date timestamp without time zone NOT NULL,  CONSTRAINT json\_buffer\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault;  --------------------- TABLE NRAVLIK ----------------------  CREATE TABLE IF NOT EXISTS public.nravlik  (  person\_id integer NOT NULL,  audio\_id integer NOT NULL,  CONSTRAINT nravlik\_pkey PRIMARY KEY (person\_id, audio\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT nravlik\_audio\_id\_fkey FOREIGN KEY (audio\_id)  REFERENCES public.audio (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT nravlik\_person\_id\_fkey FOREIGN KEY (person\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault;  --------------------- TABLE NRAVLIK\_ALBUMS ----------------------  CREATE TABLE IF NOT EXISTS public.person  (  id integer NOT NULL DEFAULT 'nextval('person\_id\_seq'::regclass)',  username character varying(32) COLLATE pg\_catalog."default" NOT NULL,  password character varying(60) COLLATE pg\_catalog."default" NOT NULL,  email character varying(256) COLLATE pg\_catalog."default" NOT NULL,  registration\_date timestamp without time zone NOT NULL,  CONSTRAINT person\_pkey PRIMARY KEY (id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT person\_email\_key UNIQUE (email)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT person\_username\_key UNIQUE (username)  USING INDEX TABLESPACE musicservicedefault  ) TABLESPACE musicservicedefault;  --------------------- TABLE PLAYLIST\_AUDIO ----------------------  CREATE TABLE IF NOT EXISTS public.playlist\_audio  (  playlist\_id integer NOT NULL,  audio\_id integer NOT NULL,  CONSTRAINT playlist\_audio\_pkey PRIMARY KEY (playlist\_id, audio\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT playlist\_audio\_audio\_id\_fkey FOREIGN KEY (audio\_id)  REFERENCES public.audio (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT playlist\_audio\_playlist\_id\_fkey FOREIGN KEY (playlist\_id)  REFERENCES public.playlist (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault;  --------------------- TABLE PLAYLIST\_AUTHOR ----------------------  CREATE TABLE IF NOT EXISTS public.playlist\_author  (  playlist\_id integer NOT NULL,  author\_id integer NOT NULL,  CONSTRAINT playlist\_author\_pkey PRIMARY KEY (playlist\_id, author\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT playlist\_author\_author\_id\_fkey FOREIGN KEY (author\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT playlist\_author\_playlist\_id\_fkey FOREIGN KEY (playlist\_id)  REFERENCES public.playlist (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault;  --------------------- TABLE ROLE\_PERSON ----------------------  CREATE TABLE IF NOT EXISTS public.role\_person  (  person\_id integer NOT NULL,  role\_id integer NOT NULL,  CONSTRAINT role\_person\_pkey PRIMARY KEY (person\_id, role\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT role\_person\_person\_id\_fkey FOREIGN KEY (person\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT role\_person\_role\_id\_fkey FOREIGN KEY (role\_id)  REFERENCES public.role (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault;  --------------------- TABLE SAVE\_AUDIO----------------------  CREATE TABLE IF NOT EXISTS public.save\_audio  (  person\_id integer NOT NULL,  audio\_id integer NOT NULL,  CONSTRAINT save\_audio\_pkey PRIMARY KEY (person\_id, audio\_id)  USING INDEX TABLESPACE musicservicedefault,  CONSTRAINT save\_audio\_audio\_id\_fkey FOREIGN KEY (audio\_id)  REFERENCES public.audio (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE,  CONSTRAINT save\_audio\_person\_id\_fkey FOREIGN KEY (person\_id)  REFERENCES public.person (id) MATCH SIMPLE  ON UPDATE NO ACTION  ON DELETE CASCADE  ) TABLESPACE musicservicedefault; |

Листинг 1 – Скрипты создания таблиц

|  |
| --- |
| -------------------------------------  ---------- CREATE ROLE --------------  -------------------------------------  CREATE ROLE Author;  CREATE ROLE Listener;  CREATE ROLE Administrator;  -------------- ADMINISTRATOR ------------------  GRANT ALL PRIVILEGES ON DATABASE "MusicDB" TO administrator;  -------------- LISTENER ------------------  GRANT SELECT, INSERT, UPDATE, DELETE ON nravlik\_albums to Listener;  GRANT SELECT, INSERT, UPDATE, DELETE ON follows to Listener;  GRANT SELECT ON album to Listener;  GRANT SELECT ON album\_audio to Listener;  GRANT SELECT ON album\_author to Listener;  GRANT SELECT ON audio to Listener;  GRANT SELECT ON author\_audio to Listener;  GRANT SELECT ON genre\_audio to Listener;  GRANT SELECT ON genre to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON save\_audio to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON nravlik to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON person to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_audio to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_author to Listener;  GRANT SELECT, INSERT, DELETE, UPDATE ON role\_person to Listener;  GRANT SELECT ON role to Listener;  GRANT SELECT, USAGE ON person\_id\_seq to Listener;  GRANT SELECT, USAGE ON genre\_id\_seq to Listener;  GRANT SELECT, USAGE ON audio\_id\_seq to Listener;  GRANT SELECT, USAGE ON playlist\_id\_seq to Listener;  GRANT SELECT, USAGE ON role\_id\_seq to Listener;  -------------- AUTHOR ------------------  GRANT SELECT, INSERT, DELETE, UPDATE ON audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON author\_audio to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON genre\_audio to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON nravlik to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON nravlik\_albums to Author;  GRANT SELECT, INSERT, UPDATE, DELETE ON follows to Author;  GRANT SELECT ON genre to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON album to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON album\_audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON album\_author to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON save\_audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON person to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_audio to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON playlist\_author to Author;  GRANT SELECT, INSERT, DELETE, UPDATE ON role\_person to Author;  GRANT SELECT ON role to Author;  GRANT SELECT, USAGE ON person\_id\_seq to Author;  GRANT SELECT ON genre\_id\_seq to Author;  GRANT SELECT, USAGE ON audio\_id\_seq to Author;  GRANT SELECT, USAGE ON playlist\_id\_seq to Author;  GRANT SELECT ON role\_id\_seq to Author;  GRANT SELECT, USAGE ON album\_id\_seq to Author; |

Листинг 2 – Скрипты создание ролей и пользователей

|  |
| --- |
| /\*---------------------------——————————----------------------------  ----------------------------| FUNCTIONS |-------------------------  -----------------------------——————————--------------------------\*/  ----------------- GET\_ALL\_AUDIOS\_BY\_AUTHOR\_NAME ----------------  CREATE OR REPLACE FUNCTION public.get\_all\_audios\_by\_author\_name(  author\_name text)  RETURNS SETOF audio  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT a.id, a.name, a.text, a.image, a.upload\_date  FROM author\_audio aa  JOIN person p ON p.id = aa.author\_id  JOIN audio a ON a.id = aa.audio\_id  WHERE (p.username = author\_name);  END;  $BODY$;  -------------------- GET\_ALL\_AUDIOS\_BY\_GENRE -------------------  CREATE OR REPLACE FUNCTION public.get\_all\_audios\_by\_genre(  \_genre\_name text)  RETURNS TABLE(audio\_id integer, audio\_name character varying, text character varying, audio\_upload\_date timestamp without time zone, authors text)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY SELECT a.id, a.name, a.text, a.upload\_date, string\_agg(p.username, ', ')  FROM genre\_audio ga  JOIN genre g ON ga.genre\_id = g.id  JOIN audio a ON ga.audio\_id = a.id  JOIN author\_audio aa on a.id = aa.audio\_id  JOIN person p on p.id = aa.author\_id  WHERE (g.genre\_name = \_genre\_name)  GROUP BY a.id;  END;  $BODY$;  ----------- GET\_ALL\_AUDIOS\_FROM\_PLAYLIST\_BY\_PLAYLIST\_ID ------------  CREATE OR REPLACE FUNCTION public.get\_all\_audios\_from\_playlist\_by\_playlist\_id(  \_playlist\_id integer)  RETURNS TABLE(audio\_name character varying, authors text)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT a.name, string\_agg(username, ', ') authors  FROM playlist\_audio paud  JOIN audio a ON a.id = paud.audio\_id  JOIN author\_audio aa ON aa.audio\_id = a.id  JOIN person p ON p.id = aa.author\_id  WHERE paud.playlist\_id = \_playlist\_id  GROUP BY a.name;  END;  $BODY$;  ----------- GET\_AMOUNT\_OF\_NRAVLICKS\_BY\_AUDIO\_ID ------------  CREATE OR REPLACE FUNCTION public.get\_amount\_of\_nravlicks\_by\_audio\_id(  \_audio\_id integer)  RETURNS integer  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  AS $BODY$  DECLARE  result INTEGER;  BEGIN  SELECT count(n.audio\_id)  INTO result  FROM nravlik n  WHERE n.audio\_id = \_audio\_id;  RETURN result;  END;  $BODY$;  ----------- GET\_AUDIO\_BY\_NAME ------------  CREATE OR REPLACE FUNCTION public.get\_audio\_by\_name(  audio\_name text)  RETURNS SETOF audio  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT \*  FROM audio a  WHERE a.name ILIKE '%' || audio\_name || '%';  END;  $BODY$;  ----------- GET\_AUDIO\_INFO\_BY\_AUDIO\_ID ------------  CREATE OR REPLACE FUNCTION public.get\_audio\_info\_by\_audio\_id(  \_audio\_id integer)  RETURNS TABLE(authors text, audio\_name character varying, upload\_date timestamp without time zone)  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT string\_agg(p.username, ', '), a.name, a.upload\_date  FROM author\_audio aa  JOIN audio a ON a.id = aa.audio\_id  JOIN person p ON p.id = aa.author\_id  WHERE aa.audio\_id = \_audio\_id  GROUP BY a.name, a.upload\_date;  END;  $BODY$;  ----------- GET\_INFO\_BY\_USERNAME ------------  CREATE OR REPLACE FUNCTION public.get\_info\_by\_username(  user\_name text)  RETURNS SETOF person  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  DECLARE  BEGIN  RETURN QUERY SELECT \*  FROM person p  WHERE (p.username = user\_name);  END;  $BODY$;  ----------- GET\_PLAYLISTS\_BY\_NAME ------------  CREATE OR REPLACE FUNCTION public.get\_playlists\_by\_name(  playlist\_name text)  RETURNS SETOF playlist  LANGUAGE 'plpgsql'  COST 100  VOLATILE PARALLEL UNSAFE  ROWS 1000  AS $BODY$  BEGIN  RETURN QUERY  SELECT \*  FROM playlist p  WHERE p.name ILIKE '%' || playlist\_name || '%';  END;  $BODY$; |

Листинг 3 – Скрипты создания функций

|  |
| --- |
| /\*--------------------------——————————----------------------------  ----------------------------| CREATE |----------------------------  ----------------------------——————————--------------------------\*/  CREATE OR REPLACE PROCEDURE public.change\_album(  IN \_id integer,  IN \_name text,  IN \_description text,  IN \_image text,  IN \_audios text,  IN \_user\_names text,  IN \_creation\_date timestamp without time zone)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  user\_names\_array text[] = string\_to\_array(\_user\_names, ',');  audios\_array text[] = string\_to\_array(\_audios, ',');  BEGIN  UPDATE album  SET name = \_name,  description = \_description,  image = \_image,  creation\_date = \_creation\_date  WHERE id = \_id;  DELETE FROM album\_author WHERE album\_id = \_id;  FOR r IN 1..cardinality(user\_names\_array)  LOOP  INSERT INTO album\_author (album\_id, author\_id)  VALUES (\_id,  (SELECT id FROM person WHERE username = user\_names\_array[r]));  END LOOP;  DELETE FROM album\_audio WHERE album\_id = \_id;  FOR r IN 1..cardinality(audios\_array)  LOOP  INSERT INTO album\_audio (album\_id, audio\_id)  VALUES (\_id,  (SELECT id FROM audio where id = audios\_array[r]::integer));  END LOOP;  END;  $BODY$;  ALTER PROCEDURE public.change\_album(integer, text, text, text, text, text, timestamp without time zone)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.change\_audio(  IN \_id integer,  IN \_name text,  IN \_text text,  IN \_image text,  IN user\_names text,  IN genres text,  IN \_upload\_date timestamp without time zone)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  res audio;  user\_names\_array text[] = string\_to\_array(user\_names, ',');  genres\_array text[] = string\_to\_array(genres, ',');  BEGIN  UPDATE audio  SET name = \_name,  text = \_text,  image = \_image,  upload\_date = \_upload\_date  WHERE id = \_id;  DELETE FROM author\_audio WHERE audio\_id = \_id;  FOR r IN 1..cardinality(user\_names\_array)  LOOP  INSERT INTO author\_audio (author\_id, audio\_id)  VALUES ((SELECT id FROM person WHERE username = user\_names\_array[r]),  \_id);  END LOOP;    DELETE FROM genre\_audio WHERE audio\_id = \_id;  FOR r IN 1..cardinality(genres\_array)  LOOP  INSERT INTO genre\_audio (genre\_id, audio\_id)  VALUES ((SELECT id FROM genre WHERE genre\_name = genres\_array[r]),  \_id);  END LOOP;  END;  $BODY$;  ALTER PROCEDURE public.change\_audio(integer, text, text, text, text, text, timestamp without time zone)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.change\_genre(  IN \_id integer,  IN \_name character varying)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  UPDATE genre  SET genre\_name = \_name  WHERE id = \_id;  END;  $BODY$;  ALTER PROCEDURE public.change\_genre(integer, character varying)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.change\_person(  IN \_lname text,  IN \_password text,  IN \_email text,  IN \_role integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  UPDATE person  SET username = \_lname,  password = md5(\_password),  email = \_email  WHERE username = \_lname;  UPDATE role\_person  SET role\_id = \_role  WHERE person\_id = (SELECT id FROM person WHERE username = \_lname);  END;  $BODY$;  ALTER PROCEDURE public.change\_person(text, text, text, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.change\_playlist(  IN \_id integer,  IN \_name text,  IN \_description text,  IN \_image text,  IN \_audios text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  audios\_array text[] = string\_to\_array(\_audios, ',');  BEGIN  UPDATE playlist  SET name = \_name,  description = \_description,  image = \_image  WHERE id = \_id;  DELETE FROM playlist\_audio WHERE playlist\_id = \_id;  FOR r IN 1..cardinality(audios\_array)  LOOP  INSERT INTO playlist\_audio (playlist\_id, audio\_id)  VALUES (\_id,  (SELECT id FROM audio where id = audios\_array[r]::integer));  END LOOP;  END;  $BODY$;  ALTER PROCEDURE public.change\_playlist(integer, text, text, text, text)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_album(  IN album\_name text,  IN album\_description text,  IN \_image text,  IN audios text,  IN authors text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  res album;  audios\_array text[] = string\_to\_array(audios, ',');  authors\_array text[] = string\_to\_array(authors, ',');  BEGIN  INSERT INTO album (name, description, creation\_date, image)  VALUES (album\_name, album\_description, now(), \_image)  RETURNING \* INTO res;  FOR r IN 1..cardinality(authors\_array)  LOOP  INSERT INTO album\_author (album\_id, author\_id)  VALUES (res.id,  (SELECT id FROM person WHERE username = authors\_array[r]));  END LOOP;  FOR r IN 1..cardinality(audios\_array)  LOOP  INSERT INTO album\_audio (album\_id, audio\_id)  VALUES (res.id,  (SELECT id FROM audio where id = audios\_array[r]::integer));  END LOOP;  END;  $BODY$;  ALTER PROCEDURE public.new\_album(text, text, text, text, text)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_audio(  IN \_name text,  IN \_text text,  IN \_image text,  IN user\_names text,  IN genres text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  res audio;  user\_names\_array text[] = string\_to\_array(user\_names, ',');  genres\_array text[] = string\_to\_array(genres, ',');  BEGIN  IF((SELECT role\_id FROM role\_person WHERE person\_id =  (SELECT id FROM person WHERE username = user\_names)) = 0)  THEN  INSERT INTO audio(name, text, image, upload\_date) VALUES (\_name, \_text, \_image, now()) RETURNING \* INTO res;  FOR r IN 1..cardinality(user\_names\_array)  LOOP  INSERT INTO author\_audio (author\_id, audio\_id)  VALUES ((SELECT id FROM person WHERE username = user\_names\_array[r]),  res.id);  END LOOP;    FOR r IN 1..cardinality(genres\_array)  LOOP  INSERT INTO genre\_audio (genre\_id, audio\_id)  VALUES ((SELECT id FROM genre WHERE genre\_name = genres\_array[r]),  res.id);  END LOOP;  END IF;  END;  $BODY$;  ALTER PROCEDURE public.new\_audio(text, text, text, text, text)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_follow(  IN \_author\_id integer,  IN \_person\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  INSERT INTO follows(person\_id, author\_id)  VALUES (\_person\_id, \_author\_id);  END;  $BODY$;  ALTER PROCEDURE public.new\_follow(integer, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_genre(  IN \_name character varying)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  INSERT INTO genre(genre\_name)  VALUES (\_name);  END;  $BODY$;  ALTER PROCEDURE public.new\_genre(character varying)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_nravlik(  IN \_audio\_id integer,  IN \_person\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  INSERT INTO nravlik(person\_id, audio\_id)  VALUES (\_person\_id, \_audio\_id);  END;  $BODY$;  ALTER PROCEDURE public.new\_nravlik(integer, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_nravlik\_album(  IN \_album\_id integer,  IN \_person\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  INSERT INTO nravlik\_albums(person\_id, album\_id)  VALUES (\_person\_id, \_album\_id);  END;  $BODY$;  ALTER PROCEDURE public.new\_nravlik\_album(integer, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_person(  IN \_lname text,  IN \_password text,  IN \_email text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  res person;  BEGIN  INSERT INTO person(username, password, email, registration\_date)  VALUES(\_lname, md5(\_password), \_email, now())  RETURNING \* INTO res;  INSERT INTO role\_person (person\_id, role\_id)  VALUES (res.id, (SELECT p.id FROM role p WHERE p.name = 'Default'));  END;  $BODY$;  ALTER PROCEDURE public.new\_person(text, text, text)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.new\_playlist(  IN playlist\_name text,  IN playlist\_description text,  IN \_image text,  IN audios text,  IN authors text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  res playlist;  audios\_array text[] = string\_to\_array(audios, ',');  BEGIN  INSERT INTO playlist (name, description, creation\_date, image)  VALUES (playlist\_name, playlist\_description, now(), \_image)  RETURNING \* INTO res;  INSERT INTO playlist\_author (playlist\_id, author\_id)  VALUES (res.id, (SELECT id FROM person WHERE username = authors));  FOR r IN 1..cardinality(audios\_array)  LOOP  INSERT INTO playlist\_audio (playlist\_id, audio\_id)  VALUES (res.id,  (SELECT a.id FROM audio a where a.id = audios\_array[r]::integer));  END LOOP;  END;  $BODY$;  ALTER PROCEDURE public.new\_playlist(text, text, text, text, text)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.save\_audio(  IN pers\_id integer,  IN aud\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  INSERT INTO save\_audio (person\_id, audio\_id)  VALUES (pers\_id, aud\_id);  END;  $BODY$;  ALTER PROCEDURE public.save\_audio(integer, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_album(  IN \_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM album  WHERE id = \_id;  DELETE FROM album\_audio  WHERE album\_id = \_id;  DELETE FROM album\_author  WHERE album\_id = \_id;  END;  $BODY$;  ALTER PROCEDURE public.delete\_album(integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_audio(  IN \_audio\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM audio  WHERE id =\_audio\_id;  DELETE FROM author\_audio  WHERE audio\_id = \_audio\_id;  DELETE FROM playlist\_audio  WHERE audio\_id = \_audio\_id;  DELETE FROM album\_audio  WHERE audio\_id = \_audio\_id;  END;  $BODY$;  ALTER PROCEDURE public.delete\_audio(integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_follow(  IN \_author\_id integer,  IN \_person\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM follows  WHERE (person\_id = \_person\_id AND author\_id = \_author\_id);  END;  $BODY$;  ALTER PROCEDURE public.delete\_follow(integer, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_genre(  IN \_name character varying)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM genre  WHERE genre\_name = \_name;  DELETE FROM genre\_audio  WHERE genre\_id = (SELECT id FROM genre WHERE genre\_name = \_name);  END;  $BODY$;  ALTER PROCEDURE public.delete\_genre(character varying)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_nravlik(  IN \_audio\_id integer,  IN \_person\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM nravlik  WHERE (person\_id = \_person\_id AND \_audio\_id = \_audio\_id);  END;  $BODY$;  ALTER PROCEDURE public.delete\_nravlik(integer, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_nravlik\_album(  IN \_album\_id integer,  IN \_person\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM nravlik\_albums  WHERE (person\_id = \_person\_id AND album\_id = \_album\_id);  END;  $BODY$;  ALTER PROCEDURE public.delete\_nravlik\_album(integer, integer)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_person(  IN user\_name text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE user\_id int;  BEGIN  SELECT person.id INTO STRICT user\_id  FROM person  WHERE username = user\_name;  DELETE FROM nravlik WHERE person\_id = user\_id;    DELETE FROM audio WHERE id IN  (SELECT audio\_id FROM author\_audio WHERE author\_id = user\_id);  DELETE FROM author\_audio WHERE author\_id = user\_id;    DELETE FROM playlist WHERE id IN  (SELECT playlist\_id FROM playlist\_author WHERE author\_id = user\_id);  DELETE FROM playlist\_audio WHERE playlist\_id IN  (SELECT playlist\_id FROM playlist\_author WHERE author\_id = user\_id);    DELETE FROM playlist\_author WHERE author\_id = user\_id;    DELETE FROM role\_person WHERE person\_id = user\_id;  DELETE FROM save\_audio WHERE person\_id = user\_id;  DELETE FROM person WHERE id = user\_id;  END;  $BODY$;  ALTER PROCEDURE public.delete\_person(text)  OWNER TO administrator;  CREATE OR REPLACE PROCEDURE public.delete\_playlist(  IN \_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM playlist  WHERE id = \_id;  DELETE FROM playlist\_audio  WHERE playlist\_id = \_id;  DELETE FROM playlist\_author  WHERE playlist\_id = \_id;  END;  $BODY$;  CREATE OR REPLACE PROCEDURE public.delete\_saved\_audio(  IN pers\_id integer,  IN aud\_id integer)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM save\_audio  WHERE (person\_id = pers\_id AND audio\_id = aud\_id);  END;  $BODY$; |

Листинг 4 – Скрипты создание функций CRUD

|  |
| --- |
| ------------------------------------  -------------- JSON ----------------  ------------------------------------  ------------- EXPORT ---------------  CREATE OR REPLACE PROCEDURE public.export\_to\_json\_tracks(  )  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  COPY (  SELECT jsonb\_agg(gt) FROM (  SELECT audio.id, audio.name, audio.text, audio.upload\_date,  audio.image, genre.genre\_name, person.username  FROM audio  INNER JOIN genre\_audio  ON audio.id = genre\_audio.audio\_id  INNER JOIN genre  ON genre\_audio.genre\_id = genre.id  INNER JOIN author\_audio  ON audio.id = author\_audio.audio\_id  INNER JOIN person  ON author\_audio.author\_id = person.id  ORDER BY audio.id  ) gt  ) TO 'C:\Program Files\PostgreSQL\15\data\ee\tracks.json';  END;  $BODY$;  ALTER PROCEDURE public.export\_to\_json\_tracks()  OWNER TO administrator;  ------------- IMPORT ---------------  CREATE OR REPLACE PROCEDURE public.import\_from\_json\_tracks(  IN path\_to\_file text)  LANGUAGE 'plpgsql'  AS $BODY$  DECLARE  json\_string jsonb;  BEGIN  json\_string := pg\_read\_file(path\_to\_file);    DELETE FROM json\_buffer;    INSERT INTO json\_buffer (id, name, text, upload\_date, image, genre\_name, username)  SELECT (obj->>'id')::numeric, obj->>'name', obj->>'text', (obj->>'upload\_date')::timestamp, obj->>'image', obj->>'genre\_name', obj->>'username'  FROM jsonb\_array\_elements(json\_string) AS obj;  INSERT INTO audio (id, name, text, upload\_date, image)  SELECT id, name, text, upload\_date, image  FROM json\_buffer;  INSERT INTO genre\_audio(audio\_id, genre\_id)  SELECT json\_buffer.id, genre.id  FROM json\_buffer  INNER JOIN genre  ON json\_buffer.genre\_name = genre.genre\_name;  INSERT INTO author\_audio(audio\_id, author\_id)  SELECT json\_buffer.id, person.id  FROM json\_buffer  INNER JOIN person  ON json\_buffer.username = person.username;  END;  $BODY$;  ALTER PROCEDURE public.import\_from\_json\_tracks(text)  OWNER TO administrator; |

Листинг 5 – Скрипты для импорта и экспорта

|  |
| --- |
| /\*---------------------————————————————---------------------  -----------------------| INSERT TABLES |-----------------  ------------------------————————————————-------------------\*/  CREATE OR REPLACE PROCEDURE public.insertcycleprocedure(  IN cycle\_count numeric)  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  FOR counter IN 1..cycle\_count  LOOP  INSERT INTO playlist(id, name, description, creation\_date)  VALUES (counter, 'stringType', 'Description', '2023-04-04');  END LOOP;  /\* INSERT INTO playlist(id, name, description, creation\_date)  VALUES  (658324, 'Jazz', 'Jazz playlis', '2023-05-04'),  (836853, 'Hip-Hop', 'Hip-Hop playlist', '2023-05-05'),  (386392, 'Pop', 'Pop playlist', '2023-05-07'),  (547283, 'Classic', 'Classic playlist', '2023-05-06'),  (174392, 'Dub', 'Dub playlist', '2023-05-08');  \*/  END;  $BODY$;  ALTER PROCEDURE public.insertcycleprocedure(numeric)  OWNER TO administrator;  -------------------DELETE TABLES -----------------  CREATE OR REPLACE PROCEDURE public.deletecycleprocedure(  )  LANGUAGE 'plpgsql'  AS $BODY$  BEGIN  DELETE FROM playlist;  END;  $BODY$;  ALTER PROCEDURE public.deletecycleprocedure()  OWNER TO administrator; |

Листинг 6 – Скрипты тестирования базы данных

|  |
| --- |
| ------------------------------------------------------  -------------------CREATE SEQUENCES ---------------------  ------------------------------------------------------  -----------------ALBUM  CREATE SEQUENCE IF NOT EXISTS public.album\_id\_seq  INCREMENT 1  START 1  MINVALUE 1  MAXVALUE 2147483647  CACHE 1  OWNED BY album.id;  ALTER SEQUENCE public.album\_id\_seq  OWNER TO administrator;  -----------------------AUDIO  CREATE SEQUENCE IF NOT EXISTS public.audio\_id\_seq  INCREMENT 1  START 1  MINVALUE 1  MAXVALUE 2147483647  CACHE 1  OWNED BY album.id;  ALTER SEQUENCE public. audio\_id\_seq  OWNER TO administrator;  --------------------------GENRE  CREATE SEQUENCE IF NOT EXISTS public.genre\_id\_seq  INCREMENT 1  START 1  MINVALUE 1  MAXVALUE 2147483647  CACHE 1  OWNED BY album.id;  ALTER SEQUENCE public.genre\_id\_seq  OWNER TO administrator;  --------------------------PERSON  CREATE SEQUENCE IF NOT EXISTS public.person\_id\_seq  INCREMENT 1  START 1  MINVALUE 1  MAXVALUE 2147483647  CACHE 1  OWNED BY album.id;  ALTER SEQUENCE public.person\_id\_seq  OWNER TO administrator;  ----------------------------PLAYLIST  CREATE SEQUENCE IF NOT EXISTS public.playlist\_id\_seq  INCREMENT 1  START 1  MINVALUE 1  MAXVALUE 2147483647  CACHE 1  OWNED BY album.id;  ALTER SEQUENCE public.playlist\_id\_seq  OWNER TO administrator;  --------------------------------ROLE  CREATE SEQUENCE IF NOT EXISTS public.role\_id\_seq  INCREMENT 1  START 1  MINVALUE 1  MAXVALUE 2147483647  CACHE 1  OWNED BY album.id;  ALTER SEQUENCE public.role\_id\_seq  OWNER TO administrator; |

Листинг 7 – Скрипты создания последовательностей