Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Базы данных (БД)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

База данных сервиса о кино

БГУИР КП 1-40 01 01 4 ПЗ

Студент: гр. 751001

Бобко А. В.

Руководитель:

асс. Фадеева Е. Е.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc58960756)

[1 Анализ предметной области 6](#_Toc58960757)

[1.1 Обзор аналогов 6](#_Toc58960758)

[1.2 Постановка задачи 8](#_Toc58960759)

[2 Реализация 9](#_Toc58960760)

[2.1 Выбор СУБД и средств разработки 9](#_Toc58960761)

[2.1.1. Типы баз данных 9](#_Toc58960762)

[2.1.2. Сведения о языке SQL 10](#_Toc58960763)

[2.1.3. Выбор СУБД 10](#_Toc58960764)

[2.1.4. Выбор средств разработки 11](#_Toc58960765)

[2.2 Особенности реализации 12](#_Toc58960766)

[2.3 Пользователи системы 13](#_Toc58960767)

[2.4 Обработка данных 14](#_Toc58960768)

[2.5 Взаимодействие приложения с базой данных 16](#_Toc58960769)

[3 Модель предметной области 18](#_Toc58960770)

[3.1 Понятие инфологической модели 18](#_Toc58960771)

[3.2 Выделение объектов предметной области 18](#_Toc58960772)

[3.3 Выделение атрибутов объектов предметной области 18](#_Toc58960773)

[3.4 Бизнес-логика 23](#_Toc58960774)

[3.5 Проверка на нормализацию 24](#_Toc58960775)

[4 Тестирование базы данных 25](#_Toc58960776)

[ЗАКЛЮЧЕНИЕ 26](#_Toc58960777)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc58960778)

[Приложение А Схема базы данных на языке SQL 28](#_Toc58960779)

ВВЕДЕНИЕ

Кинематограф появился в конце XIX века и, начиная от XX века и до наших дней, является одним из самых популярных видов искусства. Регулярно в кинотеатрах показываются премьеры новых фильмов, которые собирают огромное количество зрителей, а гонорары известных актеров могут достигать миллионы долларов. Постоянно проводятся различного рода фестивали. В 2016 году оборот киноиндустрии составил примерно 2 триллиона долларов. Только Голливуд выпускает в среднем около 100-120 фильмов в год. Количество подписчиков онлайн-кинотеатров постоянно растет, регулярно открываются новые студии. У известных стриминговых сервисов число пользователей достигает почти 200 миллионов человек.

Так как популярность кино постоянно набирает популярность, стали появляться различные киносервисы. Сейчас существует куча онлайн-кинотеатров и стриминговых сервисов, которые позволяют смотреть фильмы и сериалы в интернете. Также люди по всему миру активно интересуются новинками кино, ждут премьеры фильмов и читают новости из жизни кинозвезд. Поэтому в интернете существует множество журналов, блогов и информационных сайтов, в которых постоянно публикуются различного рода статьи, обзоры и новости кино.

Сейчас в интернете можно найти большое количество сайтов о кино, которые имеют информацию об колоссальном количестве фильмов, знаменитостей в мире кино, видео контента. Такие сайты стараются покрыть наибольшее количество существующих фильмов, а также предоставить пользователю возможность сохранять, сортировать, выставлять рейтинги и комментировать эти фильмы. Так как существует множество схожих сайтов, каждый сервис старается сделать так, чтобы пользователь захотел пользоваться именно его услугами.

Для таких сервисов необходима удобная база данных, которая сможет хранить огромные объемы информации и предоставлять пользователям необходимую им функциональность на сайте.

Целью данного курсового проекта является изучение основных принципов построения баз данных, а также проектирование реляционной базы данных, которая может использоваться в дальнейшем для создания сервиса о кино.

1. Анализ предметной области

В настоящее время на просторах интернета можно найти различные сайты и сервисы, содержащие огромные базы данных о фильмах и людях, работающих в сфере кино. Они предоставляют много различных функций, такие как оценка фильмов, бронирование и покупка билетов, разделение фильмов по папкам, получение уведомлений о новых премьерах. В ходе проектирования были проанализированы и изучены некоторые аналоги, выявлены их положительные и отрицательные стороны. Далее проведено несколько наиболее известных примеров.

* 1. Обзор аналогов

Мобильное приложение iFoodService предназначено в первую очередь для взаимодействия клиентов и ресторанов. Функционал приложения довольно обширный. Оно позволяет искать подходящую точку питания по определенным заданным критериям, таких как кухня, локация, интерьер, меню и т.д. Также пользователь может бронировать столики в ресторане, оформлять и оплачивать заказы, оставлять отзывы заведению, официантам или блюдам.

аналога будет рассмотрен один из самых известных стриминговых аудиосервисов – SoundCloud, который насчитывает более 75 миллионов зарегистрированных пользователей. Главная страница для авторизированного пользователя показана на рисунке 1.1.

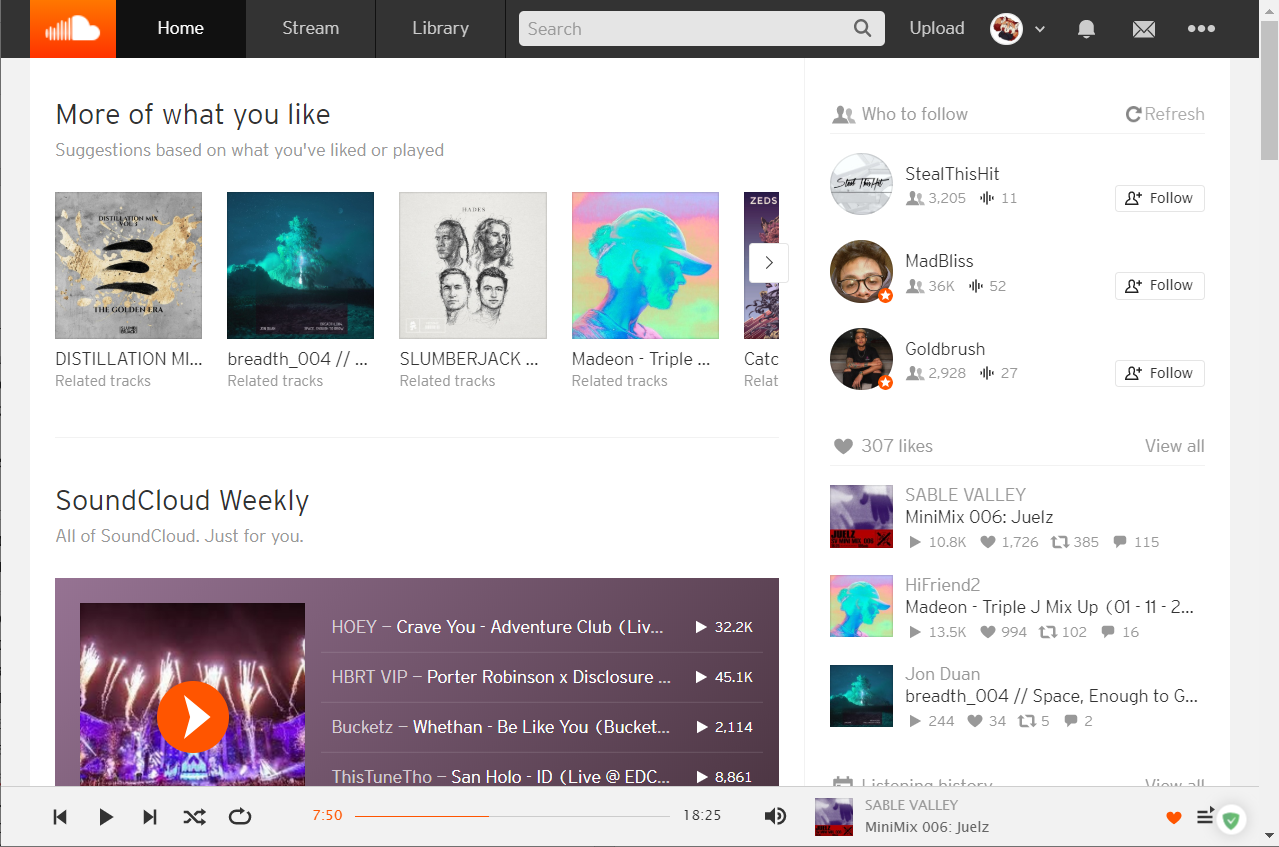


Рисунок 1.1 – главная страница авторизированного пользователя SoundCloud

На главной странице отображаются рекомендуемые композиции на основе предпочтений пользователя, недельная подборка популярных композиций, рекомендуемые для подписки пользователи и другая информация. Внизу страницы отображается плеер с воспроизводимым аудио.

Каждый пользователь SoundCloud имеет свою страницу не зависимо от того, является он исполнителем или нет. Внешний вид страницы показан на рисунке 1.2.

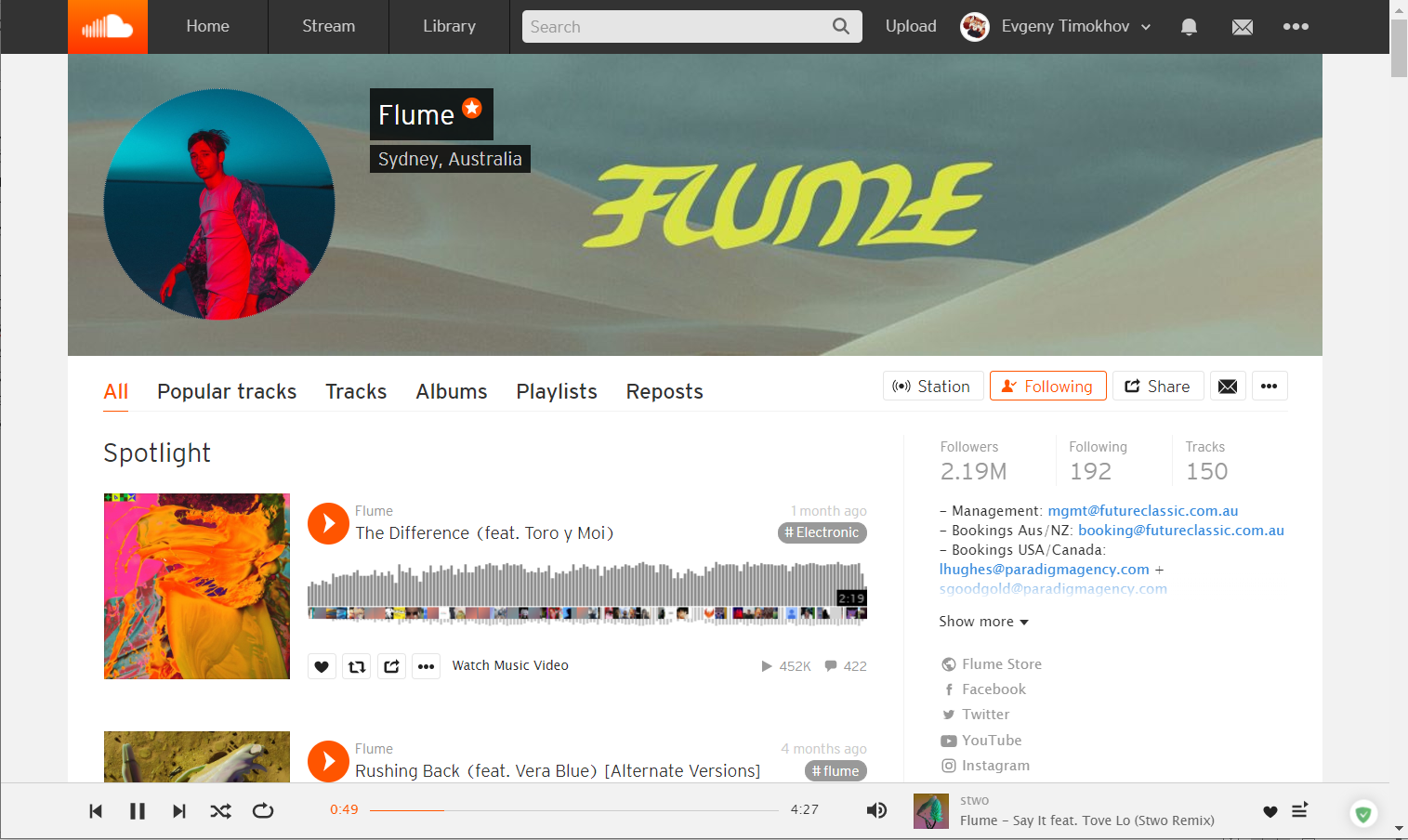


Рисунок 1.2 – страница пользователя SoundCloud

На странице любого пользователя можно увидеть его действия, собственные композиции и плейлисты, альбомы, подписки и другую информацию. Есть возможность подписаться на пользователя, чтобы видеть его обновления в своей новостной ленте.

На SoundCloud любой пользователь может создавать собственные плейлисты – наборы композиций в определенном порядке. Пример плейлиста показан на рисунке 1.3.

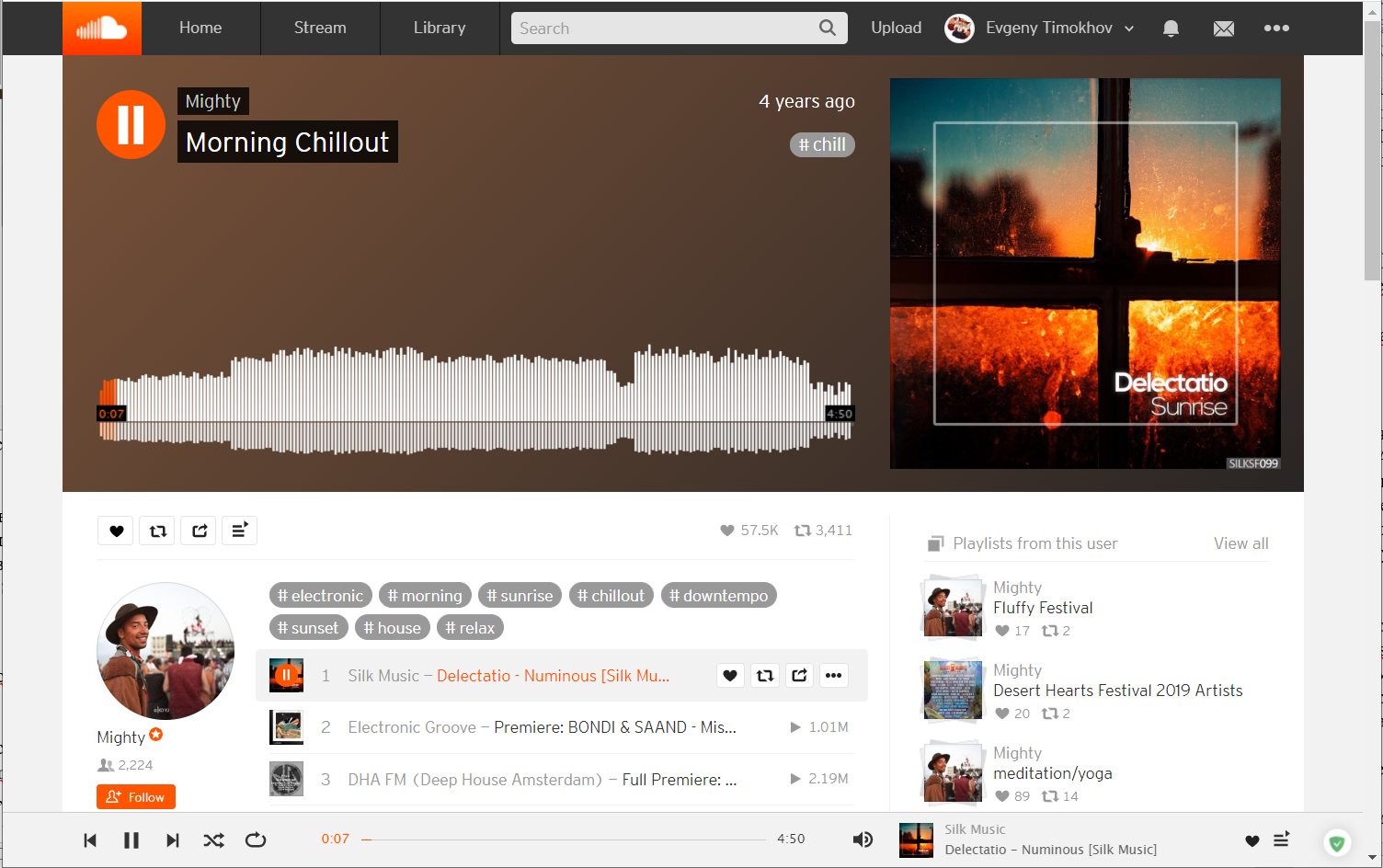


Рисунок 1.3 – плейлист на SoundCloud

В рамках данной курсовой работы планируется разработать аудиосервис с аналогичными функциями. Разработанное программное средство могло бы составить конкуренцию на рынке стриминговых аудиосервисов, таких как SoundCloud. Стриминговые сервисы при работе используют базы данных. Это могут быть как реляционные базы данных (с доступом к данным средствами языка SQL), так и нереляционные (NoSQL).

* 1. Постановка задачи

В рамках данного курсового проекта планируется разработать программное средство «Музыкальный сервис» – веб-приложение, построенное с помощью технологий Java, Spring, MySQL.

В приложении будут реализованы следующие функции:

– создание аккаунта;

– авторизация в системе;

– возможность установить изображение профиля;

– возможность загрузки своих аудиозаписей на сервис;

– считывание метаданных mp3-файла при загрузке;

– воспроизведение аудиозаписей;

– создание и воспроизведение плейлистов;

– редактирование плейлистов (добавление, удаление аудиозаписей, изменение их порядка);

– возможность оставлять комментарии к аудиозаписи;

– возможность ставить лайки аудиозаписям и плейлистам;

– возможность подписаться на других пользователей;

– отображение активности пользователей;

– статусы аккаунта с дополнительными функциями;

– возможность блокировки аккаунта пользователя администратором сервиса.

Для приложения будет спроектирована и разработана реляционная база данных, содержащая сущности и связи, отражающие предметную область музыкального сервиса (такие, как аудиозапись, альбом, плейлист, пользователь и др.), а также установлена связь между приложением и базой данных для выполнения запросов и получения их результатов.

1. Реализация
   1. Выбор СУБД и средств разработки
      1. Типы баз данных

Существуют различные типы баз данных. Выбор наилучшей базы данных для конкретного приложения зависит от множества факторов.

– Реляционные базы данных. Элементы в реляционной базе данных организованы в виде набора таблиц со столбцами и строками. Технология реляционных баз данных обеспечивает наиболее эффективный и гибкий способ доступа к структурированной информации.

– Объектно-ориентированные базы данных. Информация в объектно-ориентированной базе данных представлена в форме объекта, как в объектно-ориентированном программировании.

– Распределенные базы данных. Распределенная база данных состоит из двух или более файлов, расположенных на разных узлах. Такая база данных может храниться на нескольких компьютерах, расположенных в одном физическом месте или распределенных по разным сетям.

– Базы данных NoSQL. База данных NoSQL, или нереляционная база данных, позволяет хранить и обрабатывать неструктурированные или слабоструктурированные данные (в отличие от реляционной базы данных, задающей структуру содержащихся в ней данных). Популярность баз данных NoSQL растет по мере распространения и усложнения веб-приложений.

Для данного программного средства сделан выбор в пользу реляционной базы данных, так она имеет следующие преимущества:

– простота и доступность для понимания конечным пользователем. Единственной информационной конструкцией является таблица.

– при проектировании реляционных баз данных применяются строгие правила, базирующиеся на математическом аппарате.

– реляционная модель обеспечивает полную независимость данных. При изменении структуры реляционной базы данных изменения, которые требуется произвести в прикладных программах, как правило, минимальны.

– манипулирование данными на уровне языка СУБД производится ненавигационно, поэтому для построения запросов и написания прикладных программ нет необходимости знания конкретной организации базы данных во внешней памяти. Конечно, при исполнении запросов на физическом уровне выполняется навигация по записям таблиц, однако эти действия производятся процедурами самой СУБД.

* + 1. Сведения о языке SQL

Для манипулирования данными в СУБД используется язык SQL. SQL – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

SQL является информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

– создание в базе данных новой таблицы;

– добавление в таблицу новых записей;

– изменение записей;

– удаление записей;

– выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);

– изменение структур таблиц.

Со временем SQL усложнился – обогатился новыми конструкциями, обеспечил возможность описания и управления новыми хранимыми объектами (например, индексы, представления, триггеры и хранимые процедуры) – и стал приобретать черты, свойственные языкам программирования

При всех своих изменениях SQL остаётся самым распространённым лингвистическим средством для взаимодействия прикладного программного обеспечения с базами данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

* + 1. Выбор СУБД

В качестве реляционной системы управления базами данных выбрана MySQL. MySQL – свободная реляционная СУБД, разработку и поддержку которой осуществляет корпорация Oracle. MySQL является решением для малых и средних приложений. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

Основные преимущества MySQL:

– многопоточность, поддержка нескольких одновременных запросов;

– оптимизация связей с присоединением многих данных за один проход;

– записи фиксированной и переменной длины;

– ODBC драйвер;

– гибкая система привилегий и паролей;

– гибкая поддержка форматов чисел, строк переменной длины и меток времени;

– интерфейс с языками C и Perl, PHP;

– быстрая работа, масштабируемость;

– совместимость с ANSI SQL;

– бесплатна в большинстве случаев;

– хорошая поддержка со стороны провайдеров услуг хостинга;

– быстрая поддержка транзакций через механизм InnoDB.

* + 1. Выбор средств разработки

Наиболее важные критерии выбора технологий для разработки веб-приложений:

– независимость от платформы;

– производительность, масштабируемость;

– возможности расширения и интеграции;

– простота использования, наличие средств разработки;

– наличие необходимых программных библиотек.

Учитывая вышеперечисленные факторы выбор технологии был сделан в пользу языка Java и фреймворка Spring.

Java – сильно типизированный объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems (в последующем приобретённой компанией Oracle). Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой компьютерной архитектуре с помощью виртуальной Java-машины. На 2020 год Java – один из самых популярных языков программирования.

Программы на Java транслируются в байт-код Java, выполняемый виртуальной машиной Java (JVM) – программой, обрабатывающей байтовый код и передающей инструкции оборудованию как интерпретатор.

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять Java-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии Java является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание. Java предоставляет мощные инструменты для работы с сетью, а также технологии для разработки веб-приложений.

Spring Framework – универсальный фреймворк с открытым исходным кодом для Java-платформы. Spring обеспечивает решения многих задач, с которыми сталкиваются Java-разработчики и организации, которые хотят создать приложение на платформе Java.

Spring может быть рассмотрен как коллекция меньших фреймворков или фреймворков во фреймворке. Большинство этих фреймворков может работать независимо друг от друга, однако они обеспечивают большую функциональность при совместном их использовании. В приложении, разрабатываемом в рамках данной курсовой работы, будут использоваться следующие модули:

– Spring IoC (Inversion of Control контейнер): конфигурирование компонентов приложений и управление жизненным циклом Java-объектов;

– Spring Web MVC: каркас, основанный на HTTP и сервлетах, предоставляющий множество возможностей для расширения и кастомизации;

– Spring Security: конфигурируемый инструментарий процессов аутентификации и авторизации.

В качестве интегрированной среды разработки используется Intellij IDEA.

* 1. Особенности реализации

Для данного программного средства создана модель базы данных, обеспечивающая производительность, безопасность и гибкость архитектуры приложения.

Приложение обладает высокой производительностью, отсутствием значительных задержек при взаимодействии пользователя с сервисом, а также при взаимодействии серверной части веб-приложения с сервером базы данных. Все запросы на выборку или запись данных в таблицу выполняются за приемлемое время. Это возможно благодаря использованию СУБД MySQL, которая поддерживает многопоточность, имеет хорошую оптимизацию связей. Помимо этого, высокая производительность обеспечивается оптимальной продуманной архитектурой базы данных, использованию индексов, проведению нормализации.

При разработке программного средства и схемы базы данных значительное внимание было уделено безопасности. Аккаунт пользователя защищен паролем, при этом в базе данных хранится только его хэш. В веб-приложении используется технология Spring Security, которая предоставляет еще механизм аутентификации – защиту от CSRF-атак. Сross Site Request Forgery – межсайтовая подделка запроса – вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Помимо этого, СУБД MySQL предоставляет гибкую систему привилегий и паролей, а также высокий уровень защищенности базы данных, что позволяет предотвратить несанкционированный доступ к базе и утечку данных.

Гибкая архитектура обеспечивается использованием разделением приложения на слабосвязанные компоненты. Веб-приложение, разработанное с использованием Spring Framework, построено по архитектуре MVC, и состоит из следующих компонентов:

– представление (Thymeleaf template engine);

– модель (объекты доступа к данным, объекты сущностей, конфигурация связи с базой данных);

– контроллер (связь с представлением, слой сервисов).

Благодаря слабой связности каждый из компонентов при необходимости может быть заменен на другой, при этом изменения для других компонентов не понадобятся. Например, чтобы перейти с MySQL на другую СУБД, достаточно будет поменять конфигурацию подключения, библиотеки-драйверы, и, возможно, переписать некоторые SQL-запросы для поддержки нового диалекта выбранной СУБД. Благодаря гибкой архитектуре приложение легко расширяется при появлении новых требований.

* 1. Пользователи системы

Для аудиосервиса существует три типа пользователей – гость, зарегистрированный пользователь и администратор.

Если пользователь зашел на сайт и не прошел процедуру аутентификации (т. е. не вошел в систему и не зарегистрировался), он имеет ограниченный доступ к системе (просмотр профилей, прослушивание аудиозаписей и плейлистов). Для такого пользователя все операции, которые сервер веб-приложения посылает базе данных – это операции чтения. Гость не может изменять данные в базе посредством использования приложения. Единственное исключение – процедура регистрации нового аккаунта, которая добавит в базу новую запись в таблицу user.

Зарегистрированный пользователь имеет роль user и получает возможность изменять данные своего профиля, загружать аудиозаписи в систему, создавать и редактировать плейлисты. Для такого пользователя веб-приложение отправляет не только запросы чтения данных, но и запросы создания и редактирования записей в таблицах (создать новую запись в таблице track, изменить существующую запись в таблице playlist и т. д.).

Следующий тип пользователей – администратор. Такой пользователь имеет роли user и admin, он имеет все функции обычного пользователя, а также обладает возможностью блокировать других пользователей и изменять некоторые данные их профилей.

Роль определяется таблицей role (сущность), а соответствие пользователей ролям отражается в промежуточной таблице role\_users (связь пользователей и ролей – многие ко многим).

* 1. Обработка данных

Когда пользователи взаимодействуют с веб-приложением, его серверная часть посылает запросы в базу данных на чтение или запись информации. Входные данные для базы в большинстве случаев являются результатом действий пользователя (например, регистрация нового аккаунта, загрузка новой аудиозаписи и т. д.).

При регистрации нового аккаунта пользователь вводит данные в форму (имя пользователя, пароль, подтверждение пароля):

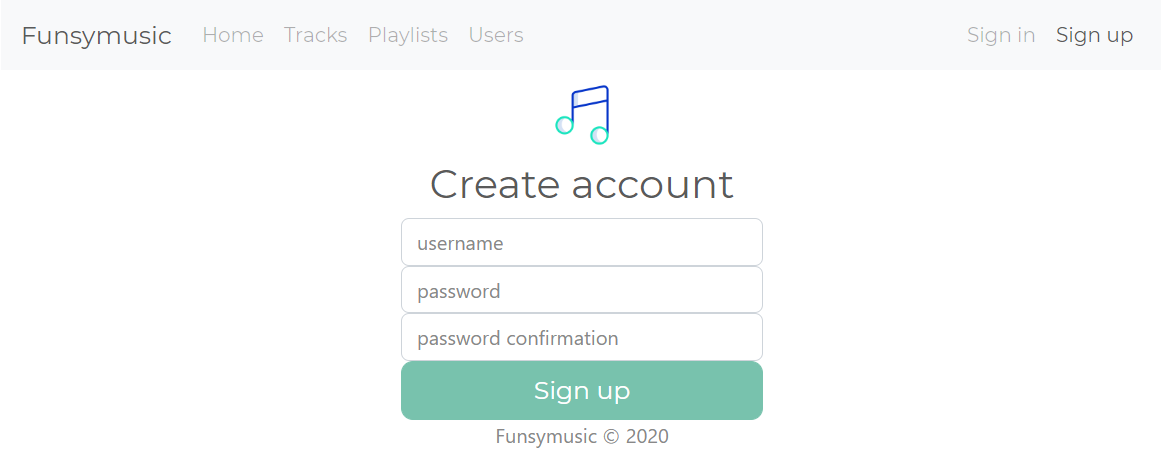


Рисунок 2.1 – форма регистрации

После нажатия кнопки “Sign up”, происходит валидация введенных пользователем данных. Если данные корректны, приложение создает объект сущности “пользователь”, устанавливает ему имя пользователя, зашифрованный пароль, роль “зарегистрированный пользователь”, дату регистрации (текущее время), а также стандартное изображение профиля. Когда объект сущности создан, приложение посылает запрос в базу данных для создания новой записи для пользователя в таблице user – в итоге пользователь зарегистрирован в системе.

В приложении существуют запросы и для выборки данных из таблицы user. Среди них – поиск пользователя по имени, который нужен для отображения данных профиля пользователя по ссылке (http://funsymusic.com/user/johndoe); выборка последних зарегистрированных в системе пользователей; выборка пользователей, на профили которых подписан текущий пользователь.

Зарегистрированные пользователи имеют возможность загружать свои аудиозаписи в систему. Для этого на специальной странице нужно выбрать файл аудиозаписи. После загрузки файла для удобства из него будут считаны метаданные (название и исполнитель). Они используются для начального заполнения формы об аудиозаписи, по желанию пользователь может их изменить.

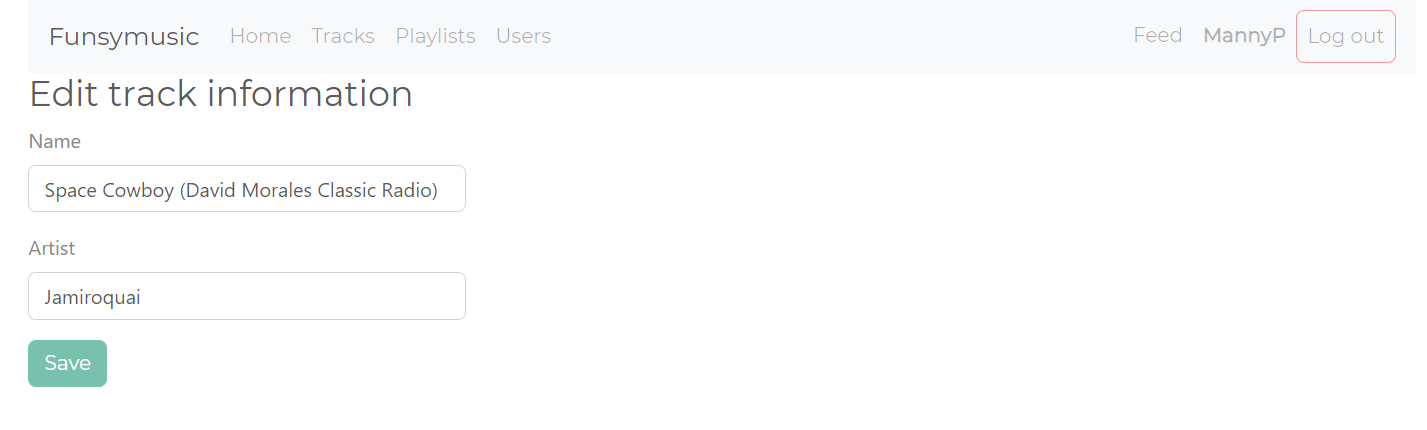


Рисунок 2.2 – автоматически заполненная данными форма редактирования информации об аудиозаписи

После нажатия кнопки “Save”, приложение создает объект сущности “аудиозапись”, заполняет его необходимыми данными (название, исполнитель, имя файла на сервере, длина аудиозаписи, автоматически считанная из метаданных файла, загрузивший пользователь, дата загрузки). После создание объекта в базу данных посылается запрос на создание новой записи в таблицу track.

Для выборки данных из таблицы track используются запросы поиска аудиозаписи по загрузившему пользователю, поиск последних загруженных в систему аудиозаписей, поиск аудиозаписей по лайкам указанного пользователя.

Приложение имеет систему подписок – пользователь может подписаться на интересующие его профили, а затем просматривать последние обновления этих профилей в своей новостной ленте. Этот механизм реализован с помощью таблицы user\_subscriptions, которая ставит пользователя в соответствие другому пользователю по признаку наличия подписки (связь многие ко многим). Когда зарегистрированный пользователь заходит на страницу профиля другого пользователя и нажимает кнопку “Subscibe”, в данную таблицу добавляется новая запись, свидетельствующая о наличии подписки.

Когда пользователь просматривает ленту активности, в ней отображаются последние события пользователей, на которых подписан текущий пользователь. Среди таких событий могут быть: загрузка новой аудиозаписи, создание нового плейлиста, оставленный комментарий, поставленный аудиозаписи или плейлисту лайк.

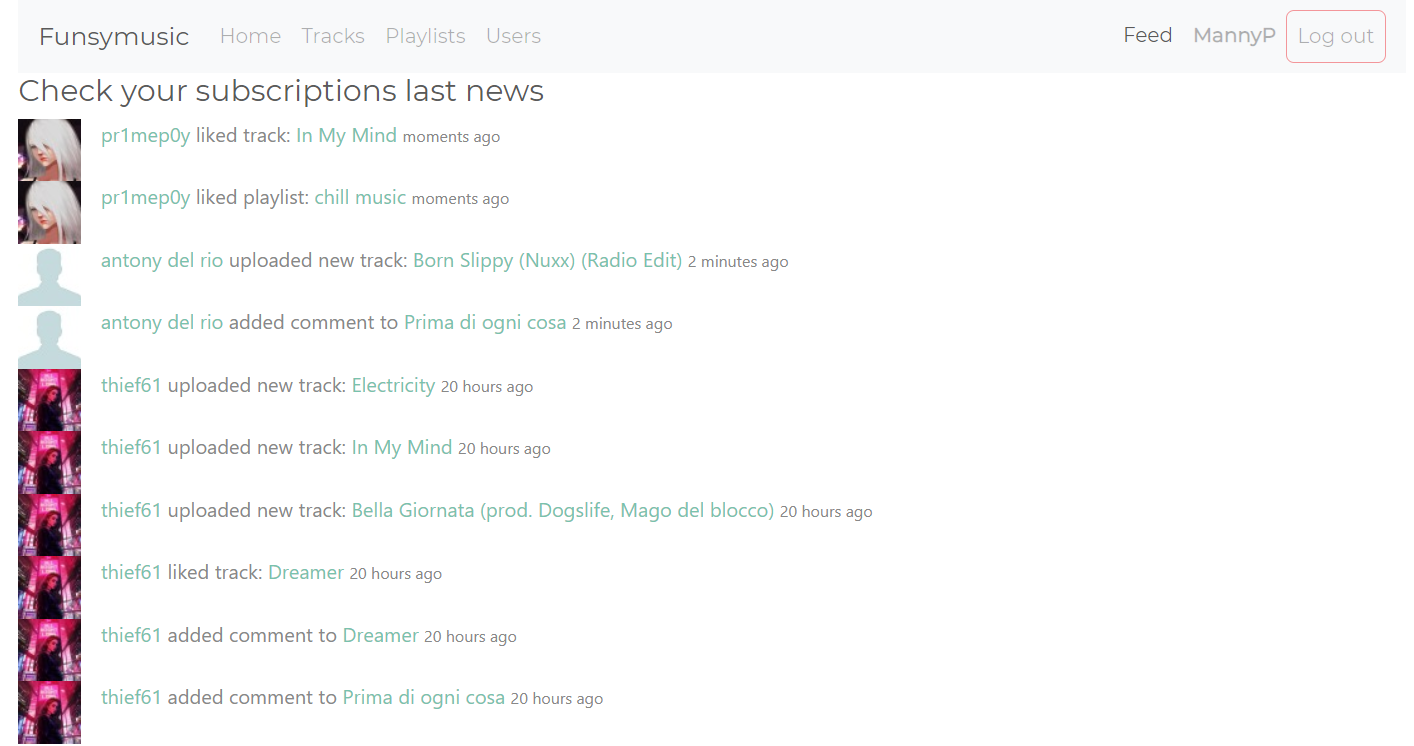


Рисунок 2.3 – лента активности

Каждое из перечисленных событий хранит свою дату в базе данных. Благодаря этому можно сделать выборку таких событий у указанных пользователей, отсортированных по дате по убыванию – в итоге получается лента активности.

Пользователи сервиса могут создавать собственные плейлисты и редактировать их. Плейлисты связаны с аудиозаписями посредством таблицы playlist\_tracks (многие ко многим). Помимо этого, таблица хранит порядковый номер аудиозаписи в плейлисте. Поэтому для отображения плейлиста не составляет труда сделать выборку аудиозаписей, относящихся к нему, отсортированных в порядке order.

* 1. Взаимодействие приложения с базой данных

Приложение взаимодействует с базой данных посредством JDBC. Это платформенно независимый промышленный стандарт взаимодействия Java-приложений с различными СУБД, реализованный в виде пакета java.sql, входящего в состав Java SE.

JDBC основан на концепции так называемых драйверов, позволяющих получать соединение с базой данных по специально описанному URL. Драйверы могут загружаться динамически (во время работы программы). Загрузившись, драйвер сам регистрирует себя и вызывается автоматически, когда программа требует URL, содержащий протокол, за который драйвер отвечает.

JDBC API содержит два основных типа интерфейсов: первый – для разработчиков приложений и второй (более низкого уровня) – для разработчиков драйверов. Соединение с базой данных описывается классом, реализующим интерфейс java.sql.Connection. Имея соединение с базой данных, можно создавать объекты типа Statement, служащие для исполнения запросов к базе данных на языке SQL.

К преимуществам JDBC можно отнести:

– лёгкость разработки: разработчик может не знать специфики базы данных, с которой работает;

– код практически не меняется, если компания переходит на другую базу данных (количество изменений зависит исключительно от различий между диалектами SQL);

– не нужно устанавливать громоздкую клиентскую программу;

– к любой базе можно подсоединиться через легко описываемый URL.

Для соединения JDBC и MySQL используется официальный драйвер MySQL Connector/J.

1. Модель предметной области
   1. Понятие инфологической модели

Инфологическая модель – ориентированная на человека и не зависимая от типа СУБД модель предметной области, определяющая совокупности информационных объектов, их атрибутов и отношений между объектами, динамику изменений предметной области, а также характер информационных потребностей пользователей.

Цель инфологического моделирования – обеспечение наиболее естественных для человека способов сбора и представления той информации, которую предполагается хранить в создаваемой базе данных.

* 1. Выделение объектов предметной области

Проанализировав предметную область разрабатываемой базы данных, можно выделить следующие основные объекты:

– пользователь;

– роль;

– тег;

– исполнитель;

– аудиозапись;

– плейлист;

– альбом;

– аудиозапись в плейлисте;

– аудиозапись в альбоме;

– лейбл;

– комментарий;

– лайк;

– подписка;

– тип расширенного аккаунта;

– запись о расширенном аккаунте;

– запись о блокировке пользователя.

* 1. Выделение атрибутов объектов предметной области

Атрибуты основных и вспомогательных сущностей разрабатываемой базы данных представлены в таблице 3.1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Сущность | Атрибут | Тип | Значение | Ограничения |
| 1 | user (пользователь) | id | bigint | идентификатор | первичный ключ |
| image | varchar(255) | адрес изображения |  |
| password | timestamp | хэш пароля | не пустой |
| registration\_date | varchar(255) | дата регистрации |  |
| username | varchar(255) | имя пользователя | не пустой |
| 2 | role (роль) | id | bigint | идентификатор | первичный ключ |
| name | varchar(255) | имя | уникальный |
| 3 | tag (тег) | id | bigint | идентификатор | первичный ключ |
| name | varchar(255) | имя |  |
| description | varchar(255) | описание |  |
| 4 | artist (исполнитель) | id | bigint | идентификатор | первичный ключ |
| name | varchar(255) | название |  |
| description | text | описание |  |
| image | varchar(255) | адрес изображения |  |
| 5 | track (аудиозапись) | id | bigint | идентификатор | первичный ключ |
| artist\_id | bigint | ид. исполнителя | внешний ключ |
| length | int | длина |  |
| media\_file | varchar(255) | адрес mp3-файла |  |
| name | varchar(255) | название |  |
| upload\_date | timestamp | дата загрузки |  |
| uploader\_id | bigint | ид. загрузившего пользователя | внешний ключ |
| 6 | playlist (плейлист) | id | bigint | идентификатор | первичный ключ |
| create\_date | timestamp | дата создания |  |
| name | varchar(255) | название |  |
| owner\_id | bigint | ид. создавшего пользователя | внешний ключ |
| 7 | album (альбом) | id | bigint | идентификатор | первичный ключ |
| name | varchar(255) | название |  |
| release\_date | timestamp | дата выхода |  |
| artist\_id | bigint | ид. исполнителя | внешний ключ |
| label\_id | bigint | ид. лейбла | внешний ключ |
| 8 | playlist\_tracks (аудиозапись в плейлисте) | playlist\_id | bigint | ид. плейлиста | первичный ключ, внешний ключ |
| tracks\_id | bigint | ид. аудиозаписи | первичный ключ, внешний ключ |
| tracks\_order | int | порядковый номер | не пустой |
| 9 | album\_tracks (аудиозапись в альбоме) | album\_id | bigint | ид. альбома | первичный ключ, внешний ключ |
| tracks\_id | bigint | ид. аудиозаписи | первичный ключ, внешний ключ |
| tracks\_order | int | порядковый номер | не пустой |
| 10 | track\_tag (тег аудиозаписи) | track\_id | bigint | ид. аудиозаписи | первичный ключ, внешний ключ |
| tag\_id | bigint | ид. тега | первичный ключ, внешний ключ |
| 11 | playlist\_tag (тег плейлиста) | playlist\_id | bigint | ид. плейлиста | первичный ключ, внешний ключ |
| tag\_id | bigint | ид. тега | первичный ключ, внешний ключ |
| 12 | album\_tag (тег альбома) | album\_id | bigint | ид. альбома | первичный ключ, внешний ключ |
| tag\_id | bigint | ид. тега | первичный ключ, внешний ключ |
| 13 | label (лейбл) | id | bigint | идентификатор | первичный ключ |
| name | varchar(255) | название |  |
| description | text | описание |  |
| country | varchar(255) | страна |  |
| 14 | track\_coment (комментарий к аудиозаписи) | id | bigint | идентификатор | первичный ключ |
| comment\_date | timestamp | дата оставления |  |
| text | text | текст |  |
| track\_timestamp | int | временная отметка аудиозаписи |  |
| track\_id | bigint | ид. аудиозаписи | внешний ключ, не пустой |
| user\_id | bigint | ид. пользователя | внешний ключ, не пустой |
| 15 | track\_like (лайк аудиозаписи) | id | bigint | bigint | первичный ключ |
| like\_date | timestamp | дата |  |
| track\_id | bigint | ид. аудиозаписи | внешний ключ, не пустой |
| user\_id | bigint | ид. пользователя | внешний ключ, не пустой |
| 16 | playlist\_like (лайк плейлиста) | id | bigint | bigint | первичный ключ |
| like\_date | timestamp | дата |  |
| playlist\_id | bigint | ид. плейлиста | внешний ключ, не пустой |
| user\_id | bigint | ид. пользователя | внешний ключ, не пустой |
| 17 | album\_like (лайк плейлиста) | id | bigint | bigint | первичный ключ |
| like\_date | timestamp | дата |  |
| album\_id | bigint | ид. альбома | внешний ключ, не пустой |
| user\_id | bigint | ид. пользователя | внешний ключ, не пустой |
| 18 | user\_subscriptions (подписка) | user\_id | bigint | ид. пользователя | первичный ключ, внешний ключ |
| subscriptions\_id | bigint | ид. пользователя, на которого подписан данный пользователь | первичный ключ, внешний ключ |
| 19 | user\_roles (роли пользователей) | user\_id | bigint | ид. пользователя | первичный ключ, внешний ключ |
| roles\_id | bigint | ид. роли | первичный ключ, внешний ключ |
| 20 | account\_upgrade\_type (тип расширенного аккаунта) | id | bigint | идентификатор | первичный ключ |
| description | varchar(255) | описание |  |
| name | varchar(255) | название |  |
| 21 | account\_upgrade\_record (запись о расширении аккаунта) | id | bigint | идентификатор | первичный ключ |
| start\_date | datetime(6) | дата начала |  |
| end\_date | datetime(6) | дата окончания |  |
| account\_upgrade\_type\_id | bigint | ид. типа расширения аккаунта | внешний ключ, не пустой |
| user\_id | bigint | ид. пользователя | внешний ключ, не пустой |
| 22 | blocking\_record (запись о блокировке аккаунта) | id | bigint | идентификатор | первичный ключ |
| blocked\_date | datetime(6) | дата начала блокировки |  |
| blocked\_until | datetime(6) | дата окончания блокировки |  |
| reason | varchar(255) | причина блокировки |  |
| user\_id | bigint | ид. пользователя | внешний ключ, не пустой |

Таблица 3.1 – Представление сущностей и атрибутов базы данных

* 1. Бизнес-логика

Бизнес-логика в базе данных представлена триггерами и процедурами.

Аудиозаписи в системе могут быть удалены. Такая ситуация может произойти по причине обращения правообладателя. Так как пользовательские плейлисты хранят порядковый номер аудиозаписи, при ее удалении целостность данных может быть нарушена – номер аудиозаписи в плейлисте будет пропущен. Чтобы не допустить данной ситуации, был создан триггер fix\_playlists. Он применяется при удалении записи из таблицы track. Если аудиозапись находилась в плейлистах, то после ее удаления для каждого плейлиста порядковые номера аудиозаписей следующей за удаляемой уменьшаются на 1, сохраняя корректность данных.

Аккаунты пользователей также могут быть удалены из системы. При этом комментарии, оставленные пользователями могут представлять ценность, и удалять их нежелательно. Для решения данной проблемы создан триггер set\_deleted\_user, который срабатывает при удалении аккаунта пользователя из системы. Для любых комментариев, оставленных данным пользователем, в качестве владельца устанавливается специальный пользователь с именем “DELETED”. Таким образом, после удаления пользователя, его комментарии будут сохранены.

Для системы рекомендаций может быть полезно выяснить предпочтения пользователя в музыке. В базе данных существует процедура find\_favourite\_track\_tag\_by\_playlists. Ее задача – для указанного пользователя определить любимый тег аудиозаписей. Она работает следующим образом: для всех аудиозаписей из плейлистов, созданных данным пользователем, подсчитывается число каждого тега. Самый популярный тег возвращается выходным параметром.

* 1. Проверка на нормализацию

Проверка на первую нормальную форму. В структуре базы данных отсутствуют сложные атрибуты, а также не наблюдаются многозначные атрибуты, т.е. таблицы приведены к первой нормальной форме.

Проверка на вторую нормальную форму. Просмотрев все отношения, описанные выше, можно сказать, что неключевые атрибуты этих отношений функционально полно зависят от первичных ключей, что является доказательством того, что отношения приведены ко второй нормальной форме.

Проверка на третью нормальную форму. Транзитивные функциональные зависимости отсутствуют, поэтому отношения соответствуют третьей нормальной форме.

1. Тестирование базы данных

Тестирование проводится с целью обеспечить качество разрабатываемого программного продукта. Стандарт ISO/IEC 12207-2003, посвященный описанию систем обеспечения качества программного обеспечения, под качеством понимает совокупность характеристик программного продукта, относящихся к его способности удовлетворять установленные и предполагаемые потребности клиента.

Для тестирования разработанной базы данных будут использоваться следующие запросы:

1. Добавление записи в таблицу.

INSERT INTO user (id, password, username) VALUES

(100, 'password', 'DELETED');

Ожидаемый результат: запись добавлена в таблицу.

1. Изменение записи в таблице.

UPDATE user

SET username = 'john1234'

WHERE id = 1;

Ожидаемый результат: имя пользователя с идентификатором 1 изменено на ‘john1234’.

1. Удаление записей, на которые не ссылаются записи другие из других таблиц.

DELETE FROM label WHERE id = 100;

Ожидаемый результат: запись удалится.

1. Удаление записей, на которые ссылаются записи другие из других таблиц.

DELETE FROM label WHERE id = 101;

Ожидаемый результат: удаление запрещено.

1. Выборка данных

SELECT text, comment\_date FROM track\_comment WHERE

track\_id = 5 ORDER BY comment\_date DESC;

Ожидаемый результат: вывод всех комментариев к аудиозаписи с идентификатором 5, отсортированных по дате в порядке убывания.

Для всех тестовых запросов реальный результат соответствует ожидаемому.

ЗАКЛЮЧЕНИЕ

Аудиосервисы сегодня пользуются большой популярностью. В рамках данной курсовой работы было разработано веб-приложение – аудиосервис Funsymusic. Для разработки использовалась платформа Spring, язык программирования Java и СУБД MySQL.

В процессе разработки были получены навыки создания построения веб-приложений с использованием базы данных и работы с СУБД MySQL.

Приложение реализует основные функции типичного аудиосервиса – такие, как создание аккаунта, авторизация в системе, загрузка своих аудиозаписей на сервис, воспроизведение аудиозаписей, создание, редактирование и воспроизведение плейлистов, возможность оставлять комментарии, ставить лайки, подписываться на других пользователей и просматривать активность подписок.

В приложении были реализованы следующие функции:

– создание аккаунта;

– авторизация в системе;

– возможность установить изображение профиля;

– возможность загрузки своих аудиозаписей на сервис;

– считывание метаданных mp3-файла при загрузке;

– воспроизведение аудиозаписей;

– создание и воспроизведение плейлистов;

– редактирование плейлистов (добавление, удаление аудиозаписей, изменение их порядка);

– возможность оставлять комментарии к аудиозаписи;

– возможность ставить лайки аудиозаписям и плейлистам;

– возможность подписаться на других пользователей;

– отображение активности пользователей;

– статусы аккаунта с дополнительными функциями;

– возможность блокировки аккаунта пользователя администратором сервиса.

Для приложения была спроектирована и разработана реляционная база данных, содержащая сущности и связи, отражающие предметную область музыкального сервиса, а также установлена связь между приложением и базой данных для выполнения запросов и получения их результатов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Брюс Эккель. Философия Java пер. Е. Матвеев. - 4-е полное изд. - Санкт-Петербург [и др.]: Питер, 2015. – 1165 с.
2. Г. Шилдт. Java 8. Полное руководство. 9-е полное издание. — М.: Вильямс, 2015. – 1377 с.
3. Документация Java SE 11 [электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.oracle.com/en/java/javase/11/docs/api/index.html>
4. Документация Spring Framework [электронный ресурс]. – Электронные данные. – Режим доступа: <https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html>
5. Справка MySQL 8.0 [электронный ресурс]. – Электронные данные. – Режим доступа: <https://dev.mysql.com/doc/refman/8.0/en/>
6. SQL – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/SQL
7. MySQL – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/MySQL
8. Реляционная СУБД – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/Реляционная СУБД
9. Нормальная форма – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/Нормальная форма

Приложение А  
Схема базы данных на языке SQL

-- MySQL dump 10.13 Distrib 8.0.22, for Win64 (x86\_64)

--

-- Host: localhost Database: funsymusicdatabase

-- ------------------------------------------------------

-- Server version 8.0.22

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;

/\*!50503 SET NAMES utf8 \*/;

/\*!40103 SET @OLD\_TIME\_ZONE=@@TIME\_ZONE \*/;

/\*!40103 SET TIME\_ZONE='+00:00' \*/;

/\*!40014 SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

/\*!40111 SET @OLD\_SQL\_NOTES=@@SQL\_NOTES, SQL\_NOTES=0 \*/;

--

-- Table structure for table `account\_upgrade\_record`

--

DROP TABLE IF EXISTS `account\_upgrade\_record`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `account\_upgrade\_record` (

`id` bigint NOT NULL,

`end\_date` datetime(6) DEFAULT NULL,

`start\_date` datetime(6) DEFAULT NULL,

`account\_upgrade\_type\_id` bigint NOT NULL,

`user\_id` bigint NOT NULL,

PRIMARY KEY (`id`),

KEY `FK\_account\_upgrade\_record\_account\_upgrade\_type` (`account\_upgrade\_type\_id`),

KEY `FK\_account\_upgrade\_record\_user` (`user\_id`),

CONSTRAINT `FK\_account\_upgrade\_record\_account\_upgrade\_type` FOREIGN KEY (`account\_upgrade\_type\_id`) REFERENCES `account\_upgrade\_type` (`id`),

CONSTRAINT `FK\_account\_upgrade\_record\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `account\_upgrade\_type`

--

DROP TABLE IF EXISTS `account\_upgrade\_type`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `account\_upgrade\_type` (

`id` bigint NOT NULL,

`description` varchar(255) DEFAULT NULL,

`name` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `album`

--

DROP TABLE IF EXISTS `album`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `album` (

`id` bigint NOT NULL,

`name` varchar(255) DEFAULT NULL,

`release\_date` timestamp NULL DEFAULT NULL,

`artist\_id` bigint DEFAULT NULL,

`label\_id` bigint DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `FK\_album\_artist` (`artist\_id`),

KEY `FK\_album\_label` (`label\_id`),

CONSTRAINT `FK\_album\_artist` FOREIGN KEY (`artist\_id`) REFERENCES `artist` (`id`),

CONSTRAINT `FK\_album\_label` FOREIGN KEY (`label\_id`) REFERENCES `label` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `album\_like`

--

DROP TABLE IF EXISTS `album\_like`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `album\_like` (

`id` bigint NOT NULL,

`like\_date` timestamp NULL DEFAULT NULL,

`album\_id` bigint NOT NULL,

`user\_id` bigint NOT NULL,

PRIMARY KEY (`id`),

KEY `FK\_album\_like\_album` (`album\_id`),

KEY `FK\_album\_like\_user` (`user\_id`),

CONSTRAINT `FK\_album\_like\_album` FOREIGN KEY (`album\_id`) REFERENCES `album` (`id`),

CONSTRAINT `FK\_album\_like\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `album\_tag`

--

DROP TABLE IF EXISTS `album\_tag`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `album\_tag` (

`album\_id` bigint NOT NULL,

`tag\_id` bigint NOT NULL,

PRIMARY KEY (`album\_id`,`tag\_id`),

KEY `FK\_album\_tag\_tag` (`tag\_id`),

CONSTRAINT `FK\_album\_tag\_album` FOREIGN KEY (`album\_id`) REFERENCES `album` (`id`),

CONSTRAINT `FK\_album\_tag\_tag` FOREIGN KEY (`tag\_id`) REFERENCES `tag` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `album\_tracks`

--

DROP TABLE IF EXISTS `album\_tracks`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `album\_tracks` (

`album\_id` bigint NOT NULL,

`tracks\_id` bigint NOT NULL,

`tracks\_order` int NOT NULL,

PRIMARY KEY (`album\_id`,`tracks\_order`),

KEY `FK\_album\_tracks\_track` (`tracks\_id`),

CONSTRAINT `FK\_album\_tracks\_album` FOREIGN KEY (`album\_id`) REFERENCES `album` (`id`),

CONSTRAINT `FK\_album\_tracks\_track` FOREIGN KEY (`tracks\_id`) REFERENCES `track` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `artist`

--

DROP TABLE IF EXISTS `artist`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `artist` (

`id` bigint NOT NULL,

`name` varchar(255) DEFAULT NULL,

`description` text,

`image` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `blocking\_record`

--

DROP TABLE IF EXISTS `blocking\_record`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `blocking\_record` (

`id` bigint NOT NULL,

`blocked\_date` datetime(6) DEFAULT NULL,

`blocked\_until` datetime(6) DEFAULT NULL,

`reason` varchar(255) DEFAULT NULL,

`user\_id` bigint NOT NULL,

PRIMARY KEY (`id`),

KEY `FK\_blocking\_record\_user` (`user\_id`),

CONSTRAINT `FK\_blocking\_record\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `label`

--

DROP TABLE IF EXISTS `label`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `label` (

`id` bigint NOT NULL,

`name` varchar(255) DEFAULT NULL,

`description` text,

`country` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `playlist`

--

DROP TABLE IF EXISTS `playlist`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `playlist` (

`id` bigint NOT NULL,

`create\_date` timestamp NULL DEFAULT NULL,

`name` varchar(255) DEFAULT NULL,

`owner\_id` bigint DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `FK\_playlist\_user` (`owner\_id`),

CONSTRAINT `FK\_playlist\_user` FOREIGN KEY (`owner\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `playlist\_like`

--

DROP TABLE IF EXISTS `playlist\_like`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `playlist\_like` (

`id` bigint NOT NULL,

`like\_date` timestamp NULL DEFAULT NULL,

`playlist\_id` bigint NOT NULL,

`user\_id` bigint NOT NULL,

PRIMARY KEY (`id`),

KEY `FK\_playlist\_like\_playlist` (`playlist\_id`),

KEY `FK\_playlist\_like\_user` (`user\_id`),

CONSTRAINT `FK\_playlist\_like\_playlist` FOREIGN KEY (`playlist\_id`) REFERENCES `playlist` (`id`),

CONSTRAINT `FK\_playlist\_like\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `playlist\_tag`

--

DROP TABLE IF EXISTS `playlist\_tag`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `playlist\_tag` (

`playlist\_id` bigint NOT NULL,

`tag\_id` bigint NOT NULL,

PRIMARY KEY (`playlist\_id`,`tag\_id`),

KEY `FK\_playlist\_tag\_tag` (`tag\_id`),

CONSTRAINT `FK\_playlist\_tag\_album` FOREIGN KEY (`playlist\_id`) REFERENCES `playlist` (`id`),

CONSTRAINT `FK\_playlist\_tag\_tag` FOREIGN KEY (`tag\_id`) REFERENCES `tag` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `playlist\_tracks`

--

DROP TABLE IF EXISTS `playlist\_tracks`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `playlist\_tracks` (

`playlist\_id` bigint NOT NULL,

`tracks\_id` bigint NOT NULL,

`tracks\_order` int NOT NULL,

PRIMARY KEY (`playlist\_id`,`tracks\_order`),

KEY `FK\_playlist\_tracks\_track` (`tracks\_id`),

CONSTRAINT `FK\_playlist\_tracks\_playlist` FOREIGN KEY (`playlist\_id`) REFERENCES `playlist` (`id`),

CONSTRAINT `FK\_playlist\_tracks\_track` FOREIGN KEY (`tracks\_id`) REFERENCES `track` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `role`

--

DROP TABLE IF EXISTS `role`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `role` (

`id` bigint NOT NULL,

`name` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`),

UNIQUE KEY `UK\_role\_name` (`name`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `tag`

--

DROP TABLE IF EXISTS `tag`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `tag` (

`id` bigint NOT NULL,

`name` varchar(255) DEFAULT NULL,

`description` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `track`

--

DROP TABLE IF EXISTS `track`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `track` (

`id` bigint NOT NULL,

`artist\_id` bigint DEFAULT NULL,

`length` int DEFAULT NULL,

`media\_file` varchar(255) DEFAULT NULL,

`name` varchar(255) DEFAULT NULL,

`upload\_date` timestamp NULL DEFAULT NULL,

`uploader\_id` bigint DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `FK\_track\_user` (`uploader\_id`),

KEY `FK\_track\_artist` (`artist\_id`),

CONSTRAINT `FK\_track\_artist` FOREIGN KEY (`artist\_id`) REFERENCES `artist` (`id`),

CONSTRAINT `FK\_track\_user` FOREIGN KEY (`uploader\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/ ;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/ ;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/ ;

/\*!50003 SET character\_set\_client = utf8mb4 \*/ ;

/\*!50003 SET character\_set\_results = utf8mb4 \*/ ;

/\*!50003 SET collation\_connection = utf8mb4\_0900\_ai\_ci \*/ ;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/ ;

/\*!50003 SET sql\_mode = 'STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION' \*/ ;

DELIMITER ;;

/\*!50003 CREATE\*/ /\*!50017 DEFINER=`root`@`localhost`\*/ /\*!50003 TRIGGER `fix\_playlists` AFTER DELETE ON `track` FOR EACH ROW BEGIN

DECLARE pl\_id,

or\_num varchar(255);

DECLARE done int DEFAULT 0;

DECLARE cur CURSOR FOR

SELECT

playlist\_id,

tracks\_order

FROM playlist\_tracks

WHERE track\_id = OLD.id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

OPEN cur;

my\_loop:

LOOP

FETCH cur INTO pl\_id, or\_num;

IF done = 1 THEN

LEAVE my\_loop;

END IF;

UPDATE playlist\_tracks

SET tracks\_order = tracks\_order - 1

WHERE playlist\_id = pl\_id

AND tracks\_order > or\_num;

END LOOP;

CLOSE cur;

SET SQL\_SAFE\_UPDATES = 0;

DELETE

FROM playlist\_tracks

WHERE track\_id = OLD.id;

SET SQL\_SAFE\_UPDATES = 1;

END \*/;;

DELIMITER ;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/ ;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/ ;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/ ;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/ ;

--

-- Table structure for table `track\_comment`

--

DROP TABLE IF EXISTS `track\_comment`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `track\_comment` (

`id` bigint NOT NULL,

`comment\_date` timestamp NULL DEFAULT NULL,

`text` text,

`track\_timestamp` int DEFAULT NULL,

`track\_id` bigint NOT NULL,

`user\_id` bigint NOT NULL,

PRIMARY KEY (`id`),

KEY `FK\_track\_comment\_track` (`track\_id`),

KEY `FK\_track\_comment\_user` (`user\_id`),

CONSTRAINT `FK\_track\_comment\_track` FOREIGN KEY (`track\_id`) REFERENCES `track` (`id`),

CONSTRAINT `FK\_track\_comment\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `track\_like`

--

DROP TABLE IF EXISTS `track\_like`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `track\_like` (

`id` bigint NOT NULL,

`like\_date` timestamp NULL DEFAULT NULL,

`track\_id` bigint NOT NULL,

`user\_id` bigint NOT NULL,

PRIMARY KEY (`id`),

KEY `FK\_track\_like\_track` (`track\_id`),

KEY `FK\_track\_like\_user` (`user\_id`),

CONSTRAINT `FK\_track\_like\_track` FOREIGN KEY (`track\_id`) REFERENCES `track` (`id`),

CONSTRAINT `FK\_track\_like\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `track\_tag`

--

DROP TABLE IF EXISTS `track\_tag`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `track\_tag` (

`track\_id` bigint NOT NULL,

`tag\_id` bigint NOT NULL,

PRIMARY KEY (`track\_id`,`tag\_id`),

KEY `FK\_track\_tag\_tag` (`tag\_id`),

CONSTRAINT `FK\_track\_tag\_tag` FOREIGN KEY (`tag\_id`) REFERENCES `tag` (`id`),

CONSTRAINT `FK\_track\_tag\_track` FOREIGN KEY (`track\_id`) REFERENCES `track` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `user`

--

DROP TABLE IF EXISTS `user`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `user` (

`id` bigint NOT NULL,

`image` varchar(255) DEFAULT NULL,

`password` varchar(255) NOT NULL,

`registration\_date` timestamp NULL DEFAULT NULL,

`username` varchar(255) NOT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/ ;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/ ;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/ ;

/\*!50003 SET character\_set\_client = utf8mb4 \*/ ;

/\*!50003 SET character\_set\_results = utf8mb4 \*/ ;

/\*!50003 SET collation\_connection = utf8mb4\_0900\_ai\_ci \*/ ;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/ ;

/\*!50003 SET sql\_mode = 'STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION' \*/ ;

DELIMITER ;;

/\*!50003 CREATE\*/ /\*!50017 DEFINER=`root`@`localhost`\*/ /\*!50003 TRIGGER `set\_deleted\_user` AFTER DELETE ON `user` FOR EACH ROW BEGIN

UPDATE track\_comment tc

SET tc.user\_id = 100

WHERE tc.user\_id = OLD.id;

END \*/;;

DELIMITER ;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/ ;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/ ;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/ ;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/ ;

--

-- Table structure for table `user\_roles`

--

DROP TABLE IF EXISTS `user\_roles`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `user\_roles` (

`user\_id` bigint NOT NULL,

`roles\_id` bigint NOT NULL,

PRIMARY KEY (`user\_id`,`roles\_id`),

KEY `FK\_user\_roles\_role` (`roles\_id`),

CONSTRAINT `FK\_user\_roles\_role` FOREIGN KEY (`roles\_id`) REFERENCES `role` (`id`),

CONSTRAINT `FK\_user\_roles\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Table structure for table `user\_subscriptions`

--

DROP TABLE IF EXISTS `user\_subscriptions`;

/\*!40101 SET @saved\_cs\_client = @@character\_set\_client \*/;

/\*!50503 SET character\_set\_client = utf8mb4 \*/;

CREATE TABLE `user\_subscriptions` (

`user\_id` bigint NOT NULL,

`subscriptions\_id` bigint NOT NULL,

PRIMARY KEY (`user\_id`,`subscriptions\_id`),

KEY `FK\_user\_subscriptions\_subsciption` (`subscriptions\_id`),

CONSTRAINT `FK\_user\_subscriptions\_subsciption` FOREIGN KEY (`subscriptions\_id`) REFERENCES `user` (`id`),

CONSTRAINT `FK\_user\_subscriptions\_user` FOREIGN KEY (`user\_id`) REFERENCES `user` (`id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;

/\*!40101 SET character\_set\_client = @saved\_cs\_client \*/;

--

-- Dumping events for database 'funsymusicdatabase'

--

--

-- Dumping routines for database 'funsymusicdatabase'

--

/\*!50003 DROP PROCEDURE IF EXISTS `find\_favourite\_track\_tag\_by\_playlists` \*/;

/\*!50003 SET @saved\_cs\_client = @@character\_set\_client \*/ ;

/\*!50003 SET @saved\_cs\_results = @@character\_set\_results \*/ ;

/\*!50003 SET @saved\_col\_connection = @@collation\_connection \*/ ;

/\*!50003 SET character\_set\_client = utf8mb4 \*/ ;

/\*!50003 SET character\_set\_results = utf8mb4 \*/ ;

/\*!50003 SET collation\_connection = utf8mb4\_0900\_ai\_ci \*/ ;

/\*!50003 SET @saved\_sql\_mode = @@sql\_mode \*/ ;

/\*!50003 SET sql\_mode = 'STRICT\_TRANS\_TABLES,NO\_ENGINE\_SUBSTITUTION' \*/ ;

DELIMITER ;;

CREATE DEFINER=`root`@`localhost` PROCEDURE `find\_favourite\_track\_tag\_by\_playlists`(IN user\_id varchar(255), OUT favourite\_tag\_id varchar(255))

BEGIN

DECLARE tag\_id\_count int;

SELECT

tt.genre\_id,

COUNT(\*) AS frequency INTO favourite\_tag\_id, tag\_id\_count

FROM track\_tag tt

WHERE tt.track\_id IN (SELECT

track.id

FROM track

JOIN playlist\_tracks pt

ON pt.track\_id = track.id

JOIN playlist

ON playlist.id = pt.playlist\_id

WHERE playlist.owner\_id = user\_id)

GROUP BY tt.tag\_id

ORDER BY frequency DESC

LIMIT 1;

END ;;

DELIMITER ;

/\*!50003 SET sql\_mode = @saved\_sql\_mode \*/ ;

/\*!50003 SET character\_set\_client = @saved\_cs\_client \*/ ;

/\*!50003 SET character\_set\_results = @saved\_cs\_results \*/ ;

/\*!50003 SET collation\_connection = @saved\_col\_connection \*/ ;

/\*!40103 SET TIME\_ZONE=@OLD\_TIME\_ZONE \*/;

/\*!40101 SET SQL\_MODE=@OLD\_SQL\_MODE \*/;

/\*!40014 SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS \*/;

/\*!40014 SET UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS \*/;

/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;

/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

/\*!40111 SET SQL\_NOTES=@OLD\_SQL\_NOTES \*/;