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 курсовому проекту

на тему

«БАЗА ДАННЫХ СТРИМИНГОВОГО СЕРВИСА ДЛЯ ПРОСЛУШИВАНИЯ МУЗЫКИ»

БГУИР КП 1-40 01 01 518 ПЗ

Студент: гр. 851003 Вайтусенок И.А.

Руководитель: Марина И.М.

Минск 2021

**СОДЕРЖАНИЕ**

[Введение 5](#_Toc58765047)

[1 Анализ предметной области 6](#_Toc58765048)

[1.1 Анализ прототипов, литературных источников 6](#_Toc58765049)

[1.2 Анализ предполагаемых пользователей и выделение функциональных требований…… 8](#_Toc58765050)

[1.3 Постановка задачи 9](#_Toc58765051)

[2 Разработка модели базы данных 10](#_Toc58765052)

[2.1 Выбор СУБД и иных средств разработки 10](#_Toc58765053)

[2.2 Разработка инфологической модели предметной области 11](#_Toc58765054)

[2.2.1 Выделение объектов предметной области 11](#_Toc58765055)

[2.2.2 Выделение атрибутов объектов предметной области 12](#_Toc58765056)

[3 Разработка бизнес-логики базы данных 14](#_Toc58765057)

[3.1 Разработка триггеров базы данных 15](#_Toc58765059)

[3.2 Разработка хранимых процедур базы данных 15](#_Toc58765060)

[3.3 Выделение индексов 16](#_Toc58765061)

[4 Тестирование, проверка работоспособности и анализ полученных результатов 17](#_Toc58765062)

[Заключение 2](#_Toc58765063)0

[Список используемой литературы 2](#_Toc58765064)1

[Приложение А Схема базы данных на языке SQL 2](#_Toc58765065)2

Введение

Базой данных (БД) называют специальным образом организованные данные, хранимые в вычислительной системе. БД создается для определенной предметной области (банк, библиотека, магазин, биржа и т.д.). Сегодня базы данных можно встретить практически везде. Их используют в медицине, на транспорте, в правоохранительных органах, в городских справочных службах, на производстве и в учебных заведениях. Базы данных могут содержать в себе различную информацию, получить которую можно в считанные секунды, нажав для этого всего лишь несколько клавиш на клавиатуре компьютера.

Разработка качественной базы данных – сложная задача, поручаемая инженерам. При разработке необходимо учитывать множество различных факторов и идти на компромиссы. Социальные сети сами по себе являются сложными веб-приложениями, на разработку которых тратится огромное количество ресурсов, и разработка базы данных – неотъемлемая и важная часть процесса, как и сама база данных – неотъемлемая часть приложения социальной сети.

На сегодняшний день широко распространены веб-сервисы для прослушивания музыки и подкастов. Данные ресурсы предоставляют огромный выбор исполнителей, доступных к прослушиванию широкой аудитории с высокой скоростью отклика.

Целью данного курсового проекта является создание базы данных стримингового сервиса для прослушивания музыки.

В качестве системы управления базами данных была выбрана обьектно-реляционная MySQL.

В настоящей пояснительной записке отражены следующие этапы написания курсового проекта:

1. Анализ предметной области. Исследование существующих аналогов и информационных потребностей пользователей, постановка задачи.
2. Разработка модели базы данных. Выбор системы управления БД и иных технических средств разработки, построение инфологической модели базы данных.
3. Разработка бизнес-логики базы данных. Описание триггеров и хранимых процедур, определение индексов.
4. Тестирование базы данных. Характеристика действий, выполненных для проведения тестирования бизнес-логики базы данных.

При выполнении всех этих этапов в итоге получится база данных, полностью удовлетворяющая цели курсового проекта.

1. Анализ предметной области
   1. Анализ прототипов, литературных источников

Анализ прототипов программных разработок, проведенный во время написания данного курсового проекта, показывает, что существует большое количество стриминговых сервисов для просмотра фильмов и сериалов.

Spotify (рус. Спотифа́й) — стриминговый сервис, позволяющий легально прослушивать музыкальные композиции, аудиокниги и подкасты, не скачивая их на устройство. Доступен в виде веб-сайта, приложений для всех операционных систем, смартфонов, смарт-устройств и медиа-систем автомобилей. На сервис приходится более 36 % мирового аудиостриминга, он доступен в большинстве стран Северной и Южной Америки, практически во всех странах Европы, некоторых странах Азии и Африка, в Австралия и Океании.



Рисунок 1.1 – Интерфейс Spotify

YouTube Music — не только предоставляет всю ту же музыку, которую вы можете получить на Spotify (в основном), но и использует искусственный интеллект для автоматического составления плейлистов по вашему вкусу. Помогает и то, что YouTube принадлежит нынешнему правителю планеты Земля - компании Google. Одна из новейших функций – это доступ ко ВСЕМ версиям песни на YouTube, включая каверы любителей и других исполнителей. В бесплатной версии есть реклама.

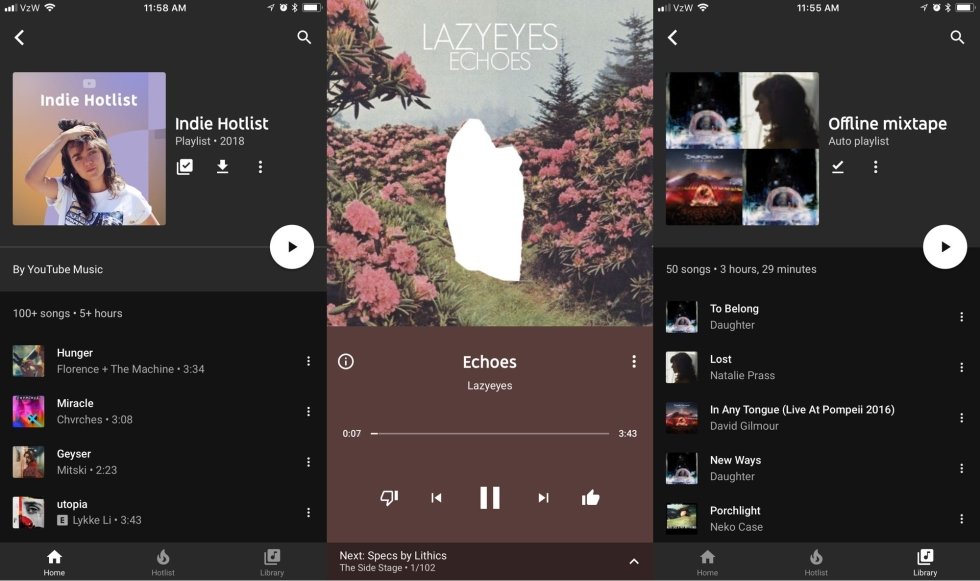


Рисунок 1.2 – Интерфейс YouTube Music

MusicUp — это одна из самых простых альтернатив Spotify, которая чрезвычайно проста в использовании. Это веб-сервис с радиостанциями в жанровом стиле, которые, по сути, являются просто плейлистами разной длины. Основная функций – плейлисты разделены на различные виды деятельности, такие как "открытая дорога", "тренировка", "расслабиться и отдохнуть" и т.д. Вы даже можете создавать свои собственные совместные комнаты с другими пользователями и голосовать за следующую песню в списке!

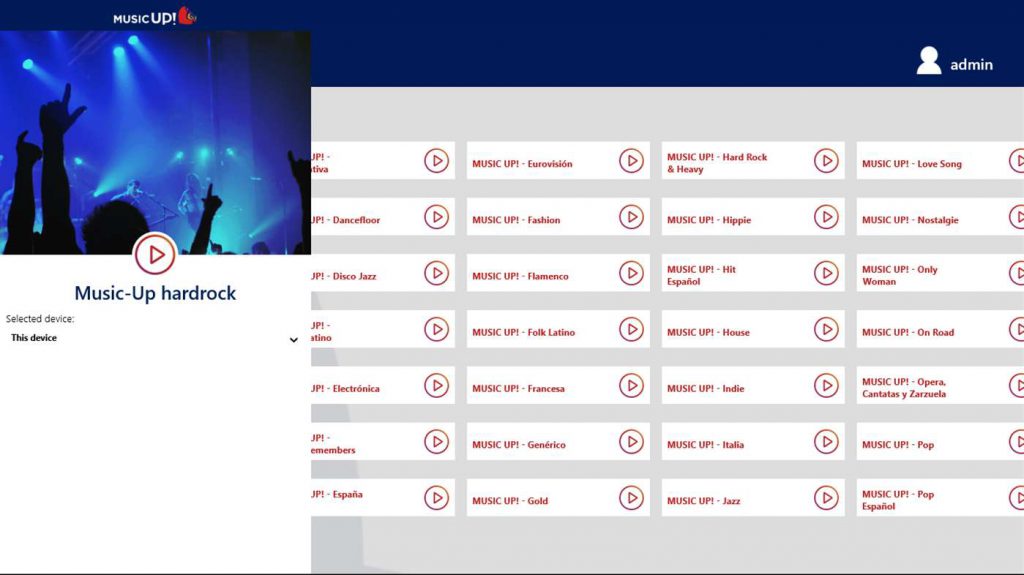


Рисунок 1.3 – Интерфейс стримингового сервиса MusicUp

* 1. Анализ предполагаемых пользователей и выделение функциональных требований

К предполагаемым пользователям относятся:

* незарегистрированный пользователь;
* зарегистрированный пользователь;

С учетом вышеописанных особенностей предметной области можно выделить следующие функциональные требования:

* регистрация пользователя;
* авторизация пользователя;
* каталог рекоммендуемых исполнителей и песен;
* работа пользователя с личным каталогом прослушанных песен;
* работа пользователя с доступными подписками;
* просмотр статистики по отдельным песням и исполнителям;
* возможность пользователя оплатить различные подписки на сервис;
* возможность пользователей увидеть историю своих платежей;
* просмотр истории специальных предложений;
* возможность просмотра биографии актеров;
* наличие служебной информации в случае ошибки;
* наличие стандартной служебной информации;
* создание плейлиста пользователем;
* комментарий пользователя под песнями, плейлистами, альбомами и исполнителями;
* возможность авторизоваться с сервисом Oauth.
* наличие информации о текущем курсе валют;
* возможность провести оплату в различных валютах;
* возможность провести оплату различными методами.
  1. Постановка задачи

Предметная область представляет собой базу данных сайта для прослушивания музыки. В данной БД хранится информация о пользователях, песнях, исполнителях, плейлистах, альбомах и доступных подписках. База данных является связующим звеном между поставщиком услуг – администратором сайта и потребителем – клиентом сайта. Также данная база данных может быть использована для работы мобильного приложения. База данных должна справляться с достаточно большим количеством подключений и обработкой данных в режиме реального времени, обеспечивать высокий уровень защиты данных и обладать достаточной гибкостью для ее изменения в будущем.

1. Разработка модели базы данных
2. Выбор СУБД и иных средств разработки

В качестве системы управления базой данных используется обьектно-реляционная MariaDB.

Преимущества MariaDB:

1. Быстродействие. Благодаря внутреннему механизму многопоточности быстродействие MariaDB весьма высоко.
2. Безопасность. Довольно высокий уровень безопасности обеспечивается благодаря базе данных mysql, создающейся при установке пакета и содержащей пять таблиц. При помощи этих таблиц можно описать, какой пользователь из какого домена с какой таблицей может работать и какие команды он может применять. Пароли, хранящиеся в базе данных, можно зашифровать при помощи встроенной в MariaDB функции password().
3. Лицензия. Раньше лицензирование MariaDB было немного запутанным; сейчас эта программа распространяется бесплатно.
4. Открытость кода. Благодаря этому вы сможете сами добавлять в пакет нужные функции, расширяя его функциональность так, как вам требуется.
5. Надежность. Этот пакет довольно стабилен и его трудно вывести из строя.
6. Сообщество. Как следствие открытости кода, бесплатности программы, стабильной и надежной ее работы образовалось сообщество людей, которые не просто лояльны к MariaDB, но и всячески участвуют как в развитии самого пакета, так и в обучении менее опытных людей работе с ним. Существует огромное количество листов рассылки и конференций, где можно получить бесплатную помощь в любое время суток.
7. Переносимость. В настоящее время существуют версии программы для большинства распространенных компьютерных платформ. Это говорит о том, что вам не навязывают определенную операционную систему. Вы сами можете выбрать, с чем работать, например с Linux или Windows, но даже в случае замены ОС вы не потеряете свои данные и вам даже не понадобятся дополнительные инструменты для их переноса.

Недостатки MariaDB:

1. Недостаточная надежность. В вопросах надежности некоторых процессов по работе с данными (например, связь, транзакции, аудит) MySQL уступает некоторым другим MariaDB.
2. Низкая скорость разработки. Как и многим другим программным продуктам с открытым кодом, MariaDB не достает некоторого технического совершенства, что порой сказывается на эффективности процессов разработки.

Так как серверная часть системы будет разрабатывается на Haskell и учитывая достаточно высокое количество запросов, требующих постоянных расчетов в реальном времени расстояния/времени/цены, многие вычисления производятся на стороне базы данных. Таким образом MariaDB выбрана в качестве базы данных благодаря быстроте, достаточной надёжности и переносимости, а также плюсом несомненно является то, что данное средство является с открытым кодом.

1. Разработка инфологической модели предметной области
   * 1. Выделение объектов предметной области

Ниже приведён список сущностей, которые будут использованы в разрабатываемой базе данных. К каждому названию сущности прикреплён краткий комментарий о назначении сущности. Также для удобства список был разделён на группы по признаку отношения к крупнейшим логическим частям БД.

1. Песня — песня и ее метаданные;
2. Аудио – файл для воспроизведения песни;
3. Похожая песня – песня, похожая на исходящуюю, используется для рекоментаций;
4. Альбом – альбом и его метаданные;
5. Альбом\_Песня – м2м связь между альбомом и песней. Потому что одна песня может быть в нескольких альбомах;
6. Артист – исполнитель и его метаданные;
7. Альбом\_Артист – м2м связь между артистом и альбомом;
8. Похожий артист;
9. Комментарий\_пользователь;
10. Комментарий\_плейлист ;
11. Комментарий\_песня ;
12. Комментарий\_альбом ;
13. Пользователь — человек, который может использовать функционал сайта;
14. Роль у пользователя – сопоставление роли и пользователя;
15. Электронная почта пользователя – логин пользователя;
16. Пароль пользователя;
17. Oauth логика для пользователя. Включает в себя три сущности;
18. Логика ошибок;
19. Обычные предупреждения;
20. Подписка;
21. Тип подписки – месяц, полгода, год;
22. Подписки пользователей – сопоставление пользователя и подписки;
23. Курс валюты;
24. Платеж ;
25. Способ платежа;
26. Статус платежа;
27. Текущий обмен валют;
28. Рекомендация;
29. Рекомендация песни, альбома или артиста;
30. Подписка на пользователя или на исполнителя;
31. Изображение у альбома, песни или пользователя;
32. Жанр;
33. Жанр песни, альбома, исполнителя;
    * 1. Выделение атрибутов объектов предметной области

Сущности и их атрибуты:

1. Песня — название песни, год ее написания, лирика песни, количество раз песня была залайкана и прослушана, ее описание
2. Аудио – место файла на диске, формат файла, его хеш;
3. Похожая песня – идентификатор песни и похожей на нее песни
4. Альбом – название альбома, количество лайков и прослушиваний, описание, студия, дата записи.;
5. Альбом\_Песня – Идентификаторы песни и альбома;
6. Артист – идентификатор артиста, его имя, является ли он премиумным пользователем, его описание и дата рождения, количество прослушиваний и лайков;
7. Альбом\_Артист – Идентификаторы артиста и альбома;
8. Похожий артист — Идентирификаторы артиста и похожего на него;
9. Комментарий\_пользователь — идентификаторы пользователя, написавшего комментарий, и пользователя, на которого странице этот комментарий был оставлен, сам текст комментария и количество лайков;
10. Комментарий\_плейлист – идентично предыдущему;
11. Комментарий\_песня – идентично предыдущему;
12. Комментарий\_альбом – идентично предыдущему;
13. Пользователь — идентификатор, имя, количество логов, возможное удаленное время;
14. Электронная почта пользователя – идентификатор, почта;
15. Пароль пользователя – соль, хеш значение пароля. Пароли в открытую не хранятся;
16. Oauth логика для пользователя – лючает в себя три сущности. Достаточно много специфичных полей, можно увидить по таблице;
17. Логика ошибок – время ошибки, ее номер и идентификатор, стек трейс;
18. Обычные предупреждения – время предупреждения, его номер и идентифиактор, строка предупреждения;
19. Подписка – идентификатор и идентификатор плана;
20. Тип подписки – идентификатор, описание, описание цены, время его работы, является ли он специальным;
21. Подписки пользователей – идентификатор пользователя и подписки и времени, когда эта подписка была совершена;
22. Курс валюты – идентификатор курса валют, два идентификатора сравниваемых валют, средняя цена, цена к концу дня;
23. Платеж – идентификатор, идентификатор плана, идентификатор валюты, статус платежа, метод платежа, время прохода платежа, количество платежей;
24. Способ платежа – идентификатор, описание, комиссия;
25. Статус платежа – идентификатор, описание, закончился платеж или нет;
26. Рекомендация – идентификатор рекомендации, идентификатор пользователя;
27. Рекомендация песни, альбома или артиста – идентификаторы песни, альбома или артиста, идентификатор рекомендации;
28. Подписка на пользователя или на исполнителя – идентификатор подписавшегося пользователя, идентификаторы подписываемого пользователя или исполнителя;
29. Изображение у альбома, песни или пользователя – адрес изображения, его идентификатор и идентификатор у альбома, песни или пользователя;
30. Жанр – идентификатр жана, его описание;
31. Разработка бизнес-логики базы данных
33. Разработка триггеров базы данных

Триггер CheckUserRegistration\_BeforeInsert срабатывает при регистрации пользователя, проверяя, не зарегистрирован ли пользователь под таким элетронным адресом, и в тоже время проверяет валидность электронного адреса.

Триггеры ValidateEmail\_BeforeInsert и ValidateEmail\_BeforeUpdate проверяют валидность почты пользователя для рассылок.

Триггеры ValidateScoreValue\_BeforeInsert и ValidateScoreValue\_BeforeUpdate проверяют значение оценки для отзыва о фильме, чтобы она была в диапазоне от 1 до 10.

Триггер UpdateUserMusic\_AfterInsert добавляет пользователю фильмы, доступные в приобретенной подписке.

Триггер DeleteUsersong\_AfterDelete удаляет фильмы у пользователя, которые были доступны по подписке.

Триггеры UpdateMusiccore\_AfterDelete, UpdateMusiccore\_AfterInsert, UpdateMusiccore\_AfterUpdate обновляют рейтинг после удаления, добавления или обновления отзыва о фильме.

Триггер CheckPreviousReview\_BeforeInsert проверяет на наличие отзыва пользователя о фильме, тем самым не давая возможности оставить более одного отзывы под одним фильмом.

Триггер UpdateShoppingCartTotal\_AfterInsert обновляет общую стоимость всех товаров, которые находятся в корзине у пользователя, при добавлении очередного товара в корзину.

1. Разработка хранимых процедур базы данных

Процедура UpdateSpecialOfferAmount обновляет количество доступных фильмов или подписок в специальном предложении.

Процедура ShowUserMusic показывает все доступные фильмы пользователя.

Процедура ShowNewMusicByCategoryForThePeriod показывает новые фильмы по категориям за выбранный период.

Процедура ShowNewUsersForTheDatePeriod показывает новых пользователей за выбранный период.

Процедура ShowLimitUsers показывает ограниченое количество пользователей в целях быстродействия.

Процедура TranslatesongCost переводит текушую стоимость фильма из одной валюты в другую на основании текущего курса валют.

Процедура TranslateSubscriptionCost переводит текущую стоимость подписки из одной валюты в другую на основании текущего курса валют.

1. Выделение индексов

Для быстрого поиска и получения данных из таблиц было решено использовать следующие индексы:

1) Artist:

* ArtistID;
* ArtistInfo (FirstName, LastName, BirthDay, Country);

2) Category:

* CategoryID;
* Name;

3) CreditCard:

* CreditCardID;
* CardTypeID;
* CardInfo (CardNumber, ExpMonth, ExpYear);

4) CreditCardType:

* CardTypeID;
* Type;

5) Currency:

* CurrencyCode;
* Name;

6) EmailAddress:

* EmailAddressID;
* UserID;
* EmailAddress;

7) ErrorLog:

* ErrorLogID;
* ErrorInfo (ErrorState, ErrorTime);

8) song:

* songID;
* OriginalLanguageID;
* Cost (Cost, CurrencyCode);
* songInfo (Title, ReleaseYear, Rating, Length, ProductionCo);

9) Language:

* LanguageID;
* Name;

10) Password:

* UserID;
* PasswordHash&Salt (PasswordHash, PasswordSalt);

11) PaymentMethod:

* PaymentMethodID;
* PaymentMethodInfo (Name, Comission);

12) Poster:

* PosterID;
* PhotoFileName (ThumbnailPhotoFileName, LargePhotoFileName);

13) Role:

* RoleID;
* Name;

14) Score:

* ScoreID;
* ScoreInfo (Value, Name);

15) SpecialOffer:

* SpecialOfferID;
* SpecialOfferInfo (StartDate, EndDate, Amount, Sale);

16) Sponsor:

* SponsorID;
* SponsorName;

17) Subscription:

* SubscriptionID, SubscriptionTypeID;
* SubscriptionInfo (Cost, CurrencyCode, DayDuration);

18) User:

* UserID;
* UserInfo (FirstName, LastName);

19) Audio:

* AudioID;
* AudioName.

1. Тестирование, проверка работоспособности и анализ полученных результатов

Таблица 4.1 – Тестирование бизнес-логики разработанной базы данных

| Номер теста | Содержание теста | Ожидаемый результат | Полученный результат | Обнаружен дефект? |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 1 | Добавление нового валидного электронного адреса пользователя в таблицу email\_address | Новый пользователь успешно добавлен в таблицу | Новый пользователь успешно добавлен в таблицу | Нет |
| 2 | Добавление нового невалидного электронного адреса пользователя в таблицу email\_address | Откат транзакции, ответ MySQL: «EmailAddress is in incorrect format!» | Откат транзакции, ответ MySQL: «EmailAddress is in incorrect format!» | Нет |
| 3 | Добавление существующего валидного электронного адреса пользователя в таблицу email\_address | Откат транзакции, ответ MySQL: «User is already registered!» | Откат транзакции, ответ MySQL: «User is already registered!» | Нет |
| 4 | Изменение электронного адреса на другой валидный у существующей записи в таблице user | Запись успешно обновлена | Запись успешно обновлена | Нет |
| 5 | Изменение электронного адреса на невалидный у существующей записи в таблице user | Откат транзакции, ответ MySQL: «EmailPromotion is in incorrect format!» | Откат транзакции, ответ MySQL: «EmailPromotion is in incorrect format!» | Нет |

Продолжение таблицы 4.1

| Номер теста | Содержание теста | Ожидаемый результат | Полученный результат | Обнаружен дефект? |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 6 | Обновление оставшегося количества у специального предложения с помощью процедуры UpdateSpecialOfferAmount | Поле Amount таблицы specialoffer для выбранного специального предложения уменьшилось на введенное количество | Поле Amount таблицы specialoffer для выбранного специального предложения уменьшилось на введенное количество | Нет |
| 7 | Добавление подписки пользователю | Подписка добавилась соответствующему пользователю в таблице user\_subscription, фильмы, вхлдящие в подписку, стали доступны пользователю и добавились в таблицу user\_song для этого пользователя | Подписка добавилась соответствующему пользователю в таблице user\_subscription, фильмы, вхлдящие в подписку, стали доступны пользователю и добавились в таблицу user\_song для этого пользователя | Нет |
| 8 | Удаление подписки у пользователя | Подписка у соответствующему пользователя удалилась из таблицы user\_subscription, фильмы, вхлдящие в подписку, стали недоступны пользователю и удалились из таблицы user\_song для этого пользователя | Подписка у соответствующему пользователя удалилась из таблицы user\_subscription, фильмы, вхлдящие в подписку, стали недоступны пользователю и удалились из таблицы user\_song для этого пользователя | Нет |
| 9 | Добавление нового отзыва о фильме пользователем | Запись добавлена в таблицу review, поле Rating в таблице song для фильма, о котором оставили отзыв, пересчиталось в соответствии с оценкой в новом отзыве | Запись добавлена в таблицу review, поле Rating в таблице song для фильма, о котором оставили отзыв, пересчиталось в соответствии с оценкой в новом отзыве | Нет |
| 10 | Удаление отзыва о фильме пользователем | Запись удалена из таблицы review, поле Rating в таблице song для фильма, о котором оставляли отзыв, пересчиталось в соответствии с оценками других пользователей | Запись удалена из таблицы review, поле Rating в таблице song для фильма, о котором оставляли отзыв, пересчиталось в соответствии с оценками других пользователей | Нет |

Продолжение таблицы 4.1

| Номер теста | Содержание теста | Ожидаемый результат | Полученный результат | Обнаружен дефект? |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| 11 | Редактирование нового отзыва о фильме пользователем | Запись обновилась в таблице review, поле Rating в таблице song для фильма, о котором оставили отзыв, пересчиталось в соответствии с оценкой в обновленном отзыве | Запись обновилась в таблице review, поле Rating в таблице song для фильма, о котором оставили отзыв, пересчиталось в соответствии с оценкой в обновленном отзыве | Нет |
| 12 | Перевод стоимости фильма из текущей валюты в выбранную на основании курса валют с помощью процедуры TranslatesongCost | SQL-запрос успешно выполнен, рассчитана стоимость фильма в введенной валюте | SQL-запрос успешно выполнен, рассчитана стоимость фильма в введенной валюте | Нет |
| 13 | Перевод стоимости подписки из текущей валюты в выбранную на основании курса валют с помощью процедуры TranslateSubscriptionCost | SQL-запрос успешно выполнен, рассчитана стоимость подписки в введенной валюте | SQL-запрос успешно выполнен, рассчитана стоимость подписки в введенной валюте | Нет |
| 14 | Добавление нового отзыва о фильме пользователем, который ранее уже оставлял свой отзыв об этом фильме | Откат транзакции, ответ MySQL: «User has already revied this song! Please, edit your existing review.» | Откат транзакции, ответ MySQL: «User has already revied this song! Please, edit your existing review.» | Нет |

После проведенного тестирования можно сделать вывод, что база данных готова к эксплуатации в типичных условиях.

Заключение

В рамках данного курсового проекта была разработана база данных стримингого сервиса для прослушивания аудиофайлов, удовлетворяющая поставленным требованиям.

Разработка была разделена на этапы для упрощения и увеличения качества проектирования. Была рассмотрена исследуемая предметная область, сформулированы требования, и только после этого было начато проектирование согласно выдвинутым требованиям, поскольку требования играют важнейшую роль при проектировании и разработки любых систем и продуктов. Проектирование также включало в себя несколько этапов различной степени абстракции (инфологический, даталогический и т.д. этапы проектирования) для большей детализации разрабатываемой БД.

При разработке были использованы некоторые приёмы, описанные в данной пояснительной записке, которые позволили удовлетворить требования по быстродействию базы данных. Также были использованы средства, позволяющие повысить надёжность и защищённость базы данных, например, триггеры, поддерживающие согласованность данных и защищающие от их случайного или намеренного повреждения и рассогласования.

База данных успешно справляется с высокой нагрузкой, обеспечивает требуемый уровень защиты данных пользователей, имеет возможность интегрироваться с системами других служб, а также является легко модифицируемой, так как предметная область предполагает постоянное добавление нового функционала и расширение существующего.

Список используемой литературы

1. Amediateka [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.amediateka.ru
2. Ivi [Электронный ресурс]. – Электронные данные. – Режим доступа: https://www.ivi.ru
3. Pinterest [Электронный ресурс]. – Электронные данные. – Режим доступа: https:/www.netflix.com
4. **Куликов, С. C. К90 Работа с MySQL, MS SQL Server и Oracle в примерах : практ. пособие. / С. С. Куликов. — Минск: БОФФ,. 2016. — 556 с.**
5. **MySQL Documentation [Электронный ресурс]. – Режим доступа:** [https://dev.mysql.com/doc](https://dev.mysql.com/doc/)
6. PostGIS Documentation **[Электронный ресурс].**  **– Режим доступа: https://postgis.net/documentation**
7. **dbdesc Official Site [Электронный ресурс]. – Режим доступа:** [https://dbmstools.com/tools/dbdesc](https://dbmstools.com/tools/dbdesc" \l "database-documentation-tools)
8. MySQL Database Service [Электронный ресурс]. – Режим доступа: https://www.mysql.com

Приложение А

(**обязательное**)

**Cхема базы данных на языке SQL**

-- MySQL Workbench Forward Engineering

SET @OLD\_UNIQUE\_CHECKS=@@UNIQUE\_CHECKS, UNIQUE\_CHECKS=0;

SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0;

SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='ONLY\_FULL\_GROUP\_BY,STRICT\_TRANS\_TABLES,NO\_ZERO\_IN\_DATE,NO\_ZERO\_DATE,ERROR\_FOR\_DIVISION\_BY\_ZERO,NO\_ENGINE\_SUBSTITUTION';

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

-- Schema Musicervicedb

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

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

-- Schema Musicervicedb

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

CREATE SCHEMA IF NOT EXISTS `Musicervicedb` DEFAULT CHARACTER SET utf8 ;

USE `Musicervicedb` ;

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

-- Table `Musicervicedb`.`Artist`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`Artist` (

`ArtistID` INT(11) NOT NULL AUTO\_INCREMENT,

`FirstName` VARCHAR(50) NOT NULL,

`LastName` VARCHAR(50) NOT NULL,

`BirthDay` DATE NOT NULL,

`Country` VARCHAR(30) CHARACTER SET 'utf8' NOT NULL,

`City` VARCHAR(30) NOT NULL,

`Biography` TEXT CHARACTER SET 'utf8' NOT NULL,

PRIMARY KEY (`ArtistID`),

INDEX `ArtistInfo` USING BTREE (`FirstName`, `LastName`, `BirthDay`, `Country`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`category`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`category` (

`CategoryID` INT(11) NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(25) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`CategoryID`),

INDEX `Name` (`Name` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`user`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`user` (

`UserID` INT(11) NOT NULL AUTO\_INCREMENT,

`FirstName` VARCHAR(50) NOT NULL,

`LastName` VARCHAR(50) NOT NULL,

`EmailPromotion` VARCHAR(50) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`UserID`),

INDEX `UserInfo` USING BTREE (`FirstName`, `LastName`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 5

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`comment`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`comment` (

`CommentID` INT(11) NOT NULL AUTO\_INCREMENT,

`UserID` INT(11) NOT NULL,

`Description` TEXT NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`CommentID`),

INDEX `UserID` (`UserID` ASC) VISIBLE,

CONSTRAINT `comment\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`creditcard\_type`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`creditcard\_type` (

`CardTypeID` INT(11) NOT NULL AUTO\_INCREMENT,

`Type` VARCHAR(20) CHARACTER SET 'utf8' NOT NULL,

PRIMARY KEY (`CardTypeID`),

INDEX `Type` (`Type` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 5

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`creditcard`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`creditcard` (

`CreditCardID` INT(11) NOT NULL AUTO\_INCREMENT,

`CardTypeID` INT(11) NOT NULL,

`CardNumber` VARCHAR(20) NOT NULL,

`ExpMonth` VARCHAR(2) NOT NULL,

`ExpYear` VARCHAR(4) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`CreditCardID`),

INDEX `CardTypeID` (`CardTypeID` ASC) VISIBLE,

INDEX `CardInfo` USING BTREE (`CardNumber`, `ExpMonth`, `ExpYear`) VISIBLE,

CONSTRAINT `creditcard\_ibfk\_1`

FOREIGN KEY (`CardTypeID`)

REFERENCES `Musicervicedb`.`creditcard\_type` (`CardTypeID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`currency`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`currency` (

`CurrencyCode` VARCHAR(3) NOT NULL,

`Name` VARCHAR(30) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY USING BTREE (`CurrencyCode`),

INDEX `Name` (`Name` ASC) VISIBLE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`currency\_rate`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`currency\_rate` (

`CurrencyRateID` INT(11) NOT NULL AUTO\_INCREMENT,

`CurrencyRateDate` DATE NOT NULL,

`FromCurrencyCode` VARCHAR(3) NOT NULL,

`ToCurrencyCode` VARCHAR(3) NOT NULL,

`AverageRate` DECIMAL(4,2) NOT NULL,

`EndOfDayRate` DECIMAL(4,2) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`CurrencyRateID`),

INDEX `FromCurrencyCode` (`FromCurrencyCode` ASC) VISIBLE,

INDEX `ToCurrencyCode` (`ToCurrencyCode` ASC) VISIBLE,

INDEX `EndOfDayRate` (`EndOfDayRate` ASC) VISIBLE,

CONSTRAINT `currency\_rate\_ibfk\_1`

FOREIGN KEY (`FromCurrencyCode`)

REFERENCES `Musicervicedb`.`currency` (`CurrencyCode`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `currency\_rate\_ibfk\_2`

FOREIGN KEY (`ToCurrencyCode`)

REFERENCES `Musicervicedb`.`currency` (`CurrencyCode`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`email\_address`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`email\_address` (

`EmailAddressID` INT(11) NOT NULL AUTO\_INCREMENT,

`UserID` INT(11) NOT NULL,

`EmailAddress` VARCHAR(50) NOT NULL,

PRIMARY KEY (`EmailAddressID`, `UserID`),

INDEX `UserID` (`UserID` ASC) VISIBLE,

INDEX `EmailAddress` (`EmailAddress` ASC) VISIBLE,

CONSTRAINT `email\_address\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 11

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`errorlog`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`errorlog` (

`ErrorLogID` INT(11) NOT NULL AUTO\_INCREMENT,

`ErrorNumber` INT(11) NOT NULL,

`ErrorState` VARCHAR(100) NOT NULL,

`ErrorMessage` VARCHAR(100) NOT NULL,

`ErrorTime` DATETIME NOT NULL,

`UserName` VARCHAR(30) NOT NULL,

PRIMARY KEY (`ErrorLogID`),

INDEX `ErrorInfo` USING BTREE (`ErrorState`, `ErrorTime`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`language`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`language` (

`LanguageID` INT(11) NOT NULL AUTO\_INCREMENT,

`Name` CHAR(30) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`LanguageID`),

INDEX `Name` (`Name` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`song`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`song` (

`songID` INT(11) NOT NULL AUTO\_INCREMENT,

`Title` VARCHAR(100) NOT NULL,

`Description` TEXT NOT NULL,

`ReleaseYear` YEAR(4) NOT NULL,

`ProductionCo` VARCHAR(100) NOT NULL,

`OriginalLanguageID` INT(11) NOT NULL,

`Cost` DECIMAL(5,2) NOT NULL,

`CurrencyCode` VARCHAR(3) NOT NULL,

`Rating` DECIMAL(4,2) NOT NULL,

`Length` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`),

INDEX `OriginalLanguageID` (`OriginalLanguageID` ASC) VISIBLE,

INDEX `song\_ibfk\_2` (`CurrencyCode` ASC) VISIBLE,

INDEX `Cost` (`Cost` ASC, `CurrencyCode` ASC) VISIBLE,

INDEX `songInfo` USING BTREE (`Title`, `ReleaseYear`, `Rating`, `Length`, `ProductionCo`) VISIBLE,

CONSTRAINT `song\_ibfk\_1`

FOREIGN KEY (`OriginalLanguageID`)

REFERENCES `Musicervicedb`.`language` (`LanguageID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `song\_ibfk\_2`

FOREIGN KEY (`CurrencyCode`)

REFERENCES `Musicervicedb`.`currency` (`CurrencyCode`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`song\_Artist`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`song\_Artist` (

`songID` INT(11) NOT NULL,

`ArtistID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`, `ArtistID`),

INDEX `ArtistID` (`ArtistID` ASC) VISIBLE,

CONSTRAINT `song\_Artist\_ibfk\_1`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `song\_Artist\_ibfk\_2`

FOREIGN KEY (`ArtistID`)

REFERENCES `Musicervicedb`.`Artist` (`ArtistID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`song\_category`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`song\_category` (

`songID` INT(11) NOT NULL,

`CategoryID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`, `CategoryID`),

INDEX `song\_category\_ibfk\_2` (`CategoryID` ASC) VISIBLE,

CONSTRAINT `song\_category\_ibfk\_1`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `song\_category\_ibfk\_2`

FOREIGN KEY (`CategoryID`)

REFERENCES `Musicervicedb`.`category` (`CategoryID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`song\_language`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`song\_language` (

`songID` INT(11) NOT NULL,

`LanguageID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`, `LanguageID`),

INDEX `LanguageID` (`LanguageID` ASC) VISIBLE,

CONSTRAINT `song\_language\_ibfk\_1`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `song\_language\_ibfk\_2`

FOREIGN KEY (`LanguageID`)

REFERENCES `Musicervicedb`.`language` (`LanguageID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`poster`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`poster` (

`PosterID` INT(11) NOT NULL AUTO\_INCREMENT,

`ThumbnailPhotoFileName` VARCHAR(100) NOT NULL,

`ThumbnailPhoto` LONGBLOB NOT NULL,

`LargePhotoFileName` VARCHAR(100) NOT NULL,

`LargePhoto` LONGBLOB NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`PosterID`),

INDEX `PhotoFileName` USING BTREE (`ThumbnailPhotoFileName`, `LargePhotoFileName`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`song\_poster`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`song\_poster` (

`songID` INT(11) NOT NULL,

`PosterID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`, `PosterID`),

INDEX `PosterID` (`PosterID` ASC) VISIBLE,

CONSTRAINT `song\_poster\_ibfk\_1`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `song\_poster\_ibfk\_2`

FOREIGN KEY (`PosterID`)

REFERENCES `Musicervicedb`.`poster` (`PosterID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`specialoffer`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`specialoffer` (

`SpecialOfferID` INT(11) NOT NULL AUTO\_INCREMENT,

`Description` TEXT NOT NULL,

`StartDate` DATE NOT NULL,

`EndDate` DATE NOT NULL,

`Amount` INT(11) NOT NULL,

`Sale` DECIMAL(4,2) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`SpecialOfferID`),

INDEX `SpecialOfferInfo` USING BTREE (`StartDate`, `EndDate`, `Amount`, `Sale`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`song\_specialoffer`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`song\_specialoffer` (

`songID` INT(11) NOT NULL,

`SpecialOfferID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`, `SpecialOfferID`),

INDEX `SpecialOfferID` (`SpecialOfferID` ASC) VISIBLE,

CONSTRAINT `song\_specialoffer\_ibfk\_1`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `song\_specialoffer\_ibfk\_2`

FOREIGN KEY (`SpecialOfferID`)

REFERENCES `Musicervicedb`.`specialoffer` (`SpecialOfferID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`Audio`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`Audio` (

`AudioID` INT(11) NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(50) NOT NULL,

`URL` VARCHAR(100) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`AudioID`),

INDEX `AudioName` USING BTREE (`Name`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`song\_Audio`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`song\_Audio` (

`songID` INT(11) NOT NULL,

`AudioID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`, `AudioID`),

INDEX `AudioID` (`AudioID` ASC) VISIBLE,

CONSTRAINT `song\_Audio\_ibfk\_1`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `song\_Audio\_ibfk\_2`

FOREIGN KEY (`AudioID`)

REFERENCES `Musicervicedb`.`Audio` (`AudioID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`password`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`password` (

`UserID` INT(11) NOT NULL,

`PasswordHash` VARCHAR(64) NOT NULL,

`PasswordSalt` VARCHAR(12) NOT NULL,

PRIMARY KEY (`UserID`),

INDEX `PasswordHash&Salt` USING BTREE (`PasswordHash`, `PasswordSalt`) VISIBLE,

CONSTRAINT `password\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8

ROW\_FORMAT = COMPACT;

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

-- Table `Musicervicedb`.`paymentmethod`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`paymentmethod` (

`PaymentMethodID` INT(11) NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(30) NOT NULL,

`Comission` DECIMAL(4,2) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`PaymentMethodID`),

INDEX `PaymentMethodInfo` USING BTREE (`Name`, `Comission`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`paymentstatus`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`paymentstatus` (

`PaymentStatusID` INT(11) NOT NULL AUTO\_INCREMENT,

`Description` VARCHAR(50) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`PaymentStatusID`))

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`shoppingcart`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`shoppingcart` (

`ShoppingCartID` INT(11) NOT NULL AUTO\_INCREMENT,

`UserID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

`TotalPrice` DECIMAL(10,2) NOT NULL,

`CurrencyCode` VARCHAR(3) NOT NULL,

PRIMARY KEY (`ShoppingCartID`, `UserID`),

INDEX `UserID` (`UserID` ASC) VISIBLE,

INDEX `TotalPrice` (`TotalPrice` ASC) VISIBLE,

INDEX `CurrencyCode` (`CurrencyCode` ASC) VISIBLE,

CONSTRAINT `shoppingcart\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `shoppingcart\_ibfk\_2`

FOREIGN KEY (`CurrencyCode`)

REFERENCES `Musicervicedb`.`currency` (`CurrencyCode`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 5

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`payment`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`payment` (

`PaymentID` INT(11) NOT NULL AUTO\_INCREMENT,

`PaymentMethodID` INT(11) NOT NULL,

`ShoppingCartID` INT(11) NOT NULL,

`Date` DATE NOT NULL,

`Amount` DECIMAL(11,2) NOT NULL,

`CurrencyCode` VARCHAR(3) NOT NULL,

`PaymentStatusID` INT(11) NOT NULL,

PRIMARY KEY (`PaymentID`),

INDEX `PaymentMethodID` (`PaymentMethodID` ASC) VISIBLE,

INDEX `PaymentStatusID` (`PaymentStatusID` ASC) VISIBLE,

INDEX `Amount` (`Amount` ASC) VISIBLE,

INDEX `ShoppingCartID` (`ShoppingCartID` ASC) VISIBLE,

INDEX `CurrencyCode` (`CurrencyCode` ASC) VISIBLE,

CONSTRAINT `payment\_ibfk\_1`

FOREIGN KEY (`PaymentMethodID`)

REFERENCES `Musicervicedb`.`paymentmethod` (`PaymentMethodID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `payment\_ibfk\_2`

FOREIGN KEY (`PaymentStatusID`)

REFERENCES `Musicervicedb`.`paymentstatus` (`PaymentStatusID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `payment\_ibfk\_3`

FOREIGN KEY (`ShoppingCartID`)

REFERENCES `Musicervicedb`.`shoppingcart` (`ShoppingCartID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `payment\_ibfk\_4`

FOREIGN KEY (`CurrencyCode`)

REFERENCES `Musicervicedb`.`currency` (`CurrencyCode`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`score`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`score` (

`ScoreID` INT(11) NOT NULL AUTO\_INCREMENT,

`Value` INT(11) NOT NULL,

`Name` VARCHAR(50) CHARACTER SET 'utf8' NOT NULL,

`Description` TEXT NOT NULL,

PRIMARY KEY (`ScoreID`),

INDEX `ScoreInfo` USING BTREE (`Value`, `Name`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 13

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`review`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`review` (

`ReviewID` INT(11) NOT NULL AUTO\_INCREMENT,

`UserID` INT(11) NOT NULL,

`songID` INT(11) NOT NULL,

`ScoreID` INT(11) NOT NULL,

`ReviewDate` DATE NOT NULL,

`Comments` TEXT NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY USING BTREE (`ReviewID`),

INDEX `UserID` (`UserID` ASC) VISIBLE,

INDEX `songID` (`songID` ASC) VISIBLE,

INDEX `ScoreID` (`ScoreID` ASC) VISIBLE,

INDEX `ReviewDate` (`ReviewDate` ASC) VISIBLE,

CONSTRAINT `review\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `review\_ibfk\_2`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `review\_ibfk\_3`

FOREIGN KEY (`ScoreID`)

REFERENCES `Musicervicedb`.`score` (`ScoreID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 15

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`review\_comment`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`review\_comment` (

`ReviewID` INT(11) NOT NULL,

`CommentID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`ReviewID`, `CommentID`),

INDEX `CommentID` (`CommentID` ASC) VISIBLE,

CONSTRAINT `review\_comment\_ibfk\_1`

FOREIGN KEY (`ReviewID`)

REFERENCES `Musicervicedb`.`review` (`ReviewID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `review\_comment\_ibfk\_2`

FOREIGN KEY (`CommentID`)

REFERENCES `Musicervicedb`.`comment` (`CommentID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`role`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`role` (

`RoleID` INT(11) NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(40) CHARACTER SET 'utf8' NOT NULL,

`Description` VARCHAR(100) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`RoleID`),

INDEX `Name` (`Name` ASC) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`shoppingcart\_item`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`shoppingcart\_item` (

`ShoppingCartItemID` INT(11) NOT NULL AUTO\_INCREMENT,

`ShoppingCartID` INT(11) NOT NULL,

`songID` INT(11) NOT NULL,

`DateCreated` DATE NOT NULL,

`ModifiedDate` DATE NOT NULL,

`Price` DECIMAL(10,2) NOT NULL,

`CurrencyCode` VARCHAR(3) NOT NULL,

PRIMARY KEY (`ShoppingCartItemID`),

INDEX `songID` (`songID` ASC) VISIBLE,

INDEX `ShoppingCartID` (`ShoppingCartID` ASC) VISIBLE,

INDEX `Price` (`Price` ASC) VISIBLE,

INDEX `CurrencyCode` (`CurrencyCode` ASC) VISIBLE,

CONSTRAINT `shoppingcart\_item\_ibfk\_1`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `shoppingcart\_item\_ibfk\_2`

FOREIGN KEY (`ShoppingCartID`)

REFERENCES `Musicervicedb`.`shoppingcart` (`ShoppingCartID`)

ON DELETE RESTRICT

ON UPDATE RESTRICT,

CONSTRAINT `shoppingcart\_item\_ibfk\_3`

FOREIGN KEY (`CurrencyCode`)

REFERENCES `Musicervicedb`.`currency` (`CurrencyCode`)

ON DELETE RESTRICT

ON UPDATE RESTRICT)

ENGINE = InnoDB

AUTO\_INCREMENT = 3

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`sponsor`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`sponsor` (

`SponsorID` INT(11) NOT NULL AUTO\_INCREMENT,

`Name` VARCHAR(20) NOT NULL,

`BusinessEmail` VARCHAR(30) NOT NULL,

`Description` TEXT NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`SponsorID`),

INDEX `SponsorName` USING BTREE (`Name`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`specialoffer\_sponsor`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`specialoffer\_sponsor` (

`SpecialOfferID` INT(11) NOT NULL,

`SponsorID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`SpecialOfferID`, `SponsorID`),

INDEX `SponsorID` (`SponsorID` ASC) VISIBLE,

CONSTRAINT `specialoffer\_sponsor\_ibfk\_1`

FOREIGN KEY (`SpecialOfferID`)

REFERENCES `Musicervicedb`.`specialoffer` (`SpecialOfferID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `specialoffer\_sponsor\_ibfk\_2`

FOREIGN KEY (`SponsorID`)

REFERENCES `Musicervicedb`.`sponsor` (`SponsorID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`subscription\_type`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`subscription\_type` (

`SubscriptionTypeID` INT(11) NOT NULL AUTO\_INCREMENT,

`Type` VARCHAR(20) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`SubscriptionTypeID`),

INDEX `SubscriptionType` USING BTREE (`Type`) VISIBLE)

ENGINE = InnoDB

AUTO\_INCREMENT = 4

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`subscription`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`subscription` (

`SubscriptionID` INT(11) NOT NULL AUTO\_INCREMENT,

`SubscriptionTypeID` INT(11) NOT NULL,

`Cost` DECIMAL(5,2) NOT NULL,

`CurrencyCode` VARCHAR(3) CHARACTER SET 'utf8' NOT NULL,

`Description` VARCHAR(100) NOT NULL,

`DayDuration` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`SubscriptionID`),

INDEX `SubscriptionTypeID` (`SubscriptionTypeID` ASC) VISIBLE,

INDEX `CurrencyCode` (`CurrencyCode` ASC) VISIBLE,

INDEX `SubscriptionInfo` USING BTREE (`Cost`, `CurrencyCode`, `DayDuration`) VISIBLE,

CONSTRAINT `subscription\_ibfk\_1`

FOREIGN KEY (`SubscriptionTypeID`)

REFERENCES `Musicervicedb`.`subscription\_type` (`SubscriptionTypeID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `subscription\_ibfk\_2`

FOREIGN KEY (`CurrencyCode`)

REFERENCES `Musicervicedb`.`currency` (`CurrencyCode`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

AUTO\_INCREMENT = 2

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`subscription\_song`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`subscription\_song` (

`SubscriptionID` INT(11) NOT NULL,

`songID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`songID`, `SubscriptionID`),

INDEX `SubscriptionID` (`SubscriptionID` ASC) VISIBLE,

CONSTRAINT `subscription\_song\_ibfk\_1`

FOREIGN KEY (`SubscriptionID`)

REFERENCES `Musicervicedb`.`subscription` (`SubscriptionID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `subscription\_song\_ibfk\_2`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`user\_creditcard`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`user\_creditcard` (

`UserID` INT(11) NOT NULL,

`CreditCardID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`UserID`, `CreditCardID`),

INDEX `CreditCardID` (`CreditCardID` ASC) VISIBLE,

CONSTRAINT `user\_creditcard\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `user\_creditcard\_ibfk\_2`

FOREIGN KEY (`CreditCardID`)

REFERENCES `Musicervicedb`.`creditcard` (`CreditCardID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`user\_song`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`user\_song` (

`UserID` INT(11) NOT NULL,

`songID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`UserID`, `songID`),

INDEX `songID` (`songID` ASC) VISIBLE,

CONSTRAINT `user\_song\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `user\_song\_ibfk\_2`

FOREIGN KEY (`songID`)

REFERENCES `Musicervicedb`.`song` (`songID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`user\_payment`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`user\_payment` (

`UserID` INT(11) NOT NULL,

`PaymentID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`UserID`, `PaymentID`),

INDEX `PaymentID` (`PaymentID` ASC) VISIBLE,

CONSTRAINT `user\_payment\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `user\_payment\_ibfk\_2`

FOREIGN KEY (`PaymentID`)

REFERENCES `Musicervicedb`.`payment` (`PaymentID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`user\_role`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`user\_role` (

`UserID` INT(11) NOT NULL,

`RoleID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`UserID`, `RoleID`),

INDEX `RoleID` (`RoleID` ASC) VISIBLE,

CONSTRAINT `user\_role\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `user\_role\_ibfk\_2`

FOREIGN KEY (`RoleID`)

REFERENCES `Musicervicedb`.`role` (`RoleID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

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

-- Table `Musicervicedb`.`user\_subscription`

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

CREATE TABLE IF NOT EXISTS `Musicervicedb`.`user\_subscription` (

`UserID` INT(11) NOT NULL,

`SubscriptionID` INT(11) NOT NULL,

`ModifiedDate` DATE NOT NULL,

PRIMARY KEY (`UserID`, `SubscriptionID`),

INDEX `SubscriptionID` (`SubscriptionID` ASC) VISIBLE,

CONSTRAINT `user\_subscription\_ibfk\_1`

FOREIGN KEY (`UserID`)

REFERENCES `Musicervicedb`.`user` (`UserID`)

ON DELETE CASCADE

ON UPDATE CASCADE,

CONSTRAINT `user\_subscription\_ibfk\_2`

FOREIGN KEY (`SubscriptionID`)

REFERENCES `Musicervicedb`.`subscription` (`SubscriptionID`)

ON DELETE CASCADE

ON UPDATE CASCADE)

ENGINE = InnoDB

DEFAULT CHARACTER SET = utf8;

USE `Musicervicedb` ;

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

-- procedure ShowLimitUsers

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

DELIMITER $$

USE `Musicervicedb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `ShowLimitUsers`(IN `UserLimit` INT)

NO SQL

BEGIN

SELECT u.FirstName, u.LastName, ea.EmailAddress, p.PasswordSalt, p.PasswordHash, u.ModifiedDate AS RegisteredDate FROM user AS u

INNER JOIN email\_address AS ea

ON ea.UserID = u.UserID

INNER JOIN password as p

ON p.UserID = u.UserID

LIMIT UserLimit;

END$$

DELIMITER ;

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

-- procedure ShowNewMusicByCategoryForThePeriod

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

DELIMITER $$

USE `Musicervicedb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `ShowNewMusicByCategoryForThePeriod`(IN `DateFrom` DATE, IN `DateTo` DATE, IN `CategoryName` VARCHAR(25) CHARSET utf8)

NO SQL

BEGIN

IF CategoryName <> ''

THEN

SELECT f.Title, c.Name, f.Rating, f.ReleaseYear, f.ProductionCo FROM song AS f

INNER JOIN song\_category as fc

ON fc.songID = f.songID

INNER JOIN category as c

ON c.CategoryID = fc.CategoryID

WHERE f.ModifiedDate >= DateFrom AND f.ModifiedDate <= DateTo AND c.Name = CategoryName;

ELSE

SELECT f.Title, c.Name, f.Rating, f.ReleaseYear, f.ProductionCo FROM song AS f

INNER JOIN song\_category as fc

ON fc.songID = f.songID

INNER JOIN category as c

ON c.CategoryID = fc.CategoryID

WHERE f.ModifiedDate >= DateFrom AND f.ModifiedDate <= DateTo;

END IF;

END$$

DELIMITER ;

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

-- procedure ShowNewUsersForTheDatePeriod

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

DELIMITER $$

USE `Musicervicedb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `ShowNewUsersForTheDatePeriod`(IN `DateFrom` DATE, IN `DateTo` DATE)

NO SQL

BEGIN

SELECT u.FirstName, u.LastName, ea.EmailAddress, p.PasswordSalt, p.PasswordHash, u.ModifiedDate AS RegisteredDate FROM user AS u

INNER JOIN email\_address AS ea

ON ea.UserID = u.UserID

INNER JOIN password as p

ON p.UserID = u.UserID

WHERE u.ModifiedDate >= DateFrom AND u.ModifiedDate <= DateTo;

END$$

DELIMITER ;

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

-- procedure ShowUserMusic

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

DELIMITER $$

USE `Musicervicedb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `ShowUserMusic`(IN `UserID` INT)

NO SQL

BEGIN

SELECT f.Title, c.Name, f.Length, f.Rating, f.ReleaseYear FROM user AS u

INNER JOIN user\_song as uf

ON u.UserID = uf.UserID

INNER JOIN song as f

ON f.songID = uf.songID

INNER JOIN song\_category as fc

ON fc.songID = f.songID

INNER JOIN category as c

ON c.CategoryID = fc.CategoryID

WHERE u.UserID = UserID;

END$$

DELIMITER ;

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

-- procedure TranslatesongCost

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

DELIMITER $$

USE `Musicervicedb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `TranslatesongCost`(IN `songID` INT, IN `NewCurrency` VARCHAR(3) CHARSET utf8, OUT `NewCurrency\_Cost` DECIMAL(5,2))

NO SQL

BEGIN

IF NewCurrency <> ''

THEN

SELECT f.Cost\*cr.AverageRate INTO NewCurrency\_Cost FROM song AS f

INNER JOIN currency\_rate AS cr

ON f.CurrencyCode = cr.FromCurrencyCode

WHERE f.songID = songID AND cr.ToCurrencyCode = NewCurrency;

ELSE

SELECT s.Cost INTO NewCurrency\_Cost FROM subscription AS s;

END IF;

END$$

DELIMITER ;

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

-- procedure TranslateSubscriptionCost

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

DELIMITER $$

USE `Musicervicedb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `TranslateSubscriptionCost`(IN `SubscriptionID` INT, IN `NewCurrency` VARCHAR(3) CHARSET utf8, OUT `NewCurrency\_Cost` DECIMAL(5,2))

NO SQL

BEGIN

IF NewCurrency <> ''

THEN

SELECT s.Cost\*cr.AverageRate INTO NewCurrency\_Cost FROM subscription AS s

INNER JOIN currency\_rate AS cr

ON s.CurrencyCode = cr.FromCurrencyCode

WHERE s.SubscriptionID = SubscriptionID AND cr.ToCurrencyCode = NewCurrency;

ELSE

SELECT s.Cost INTO NewCurrency\_Cost FROM subscription AS s;

END IF;

END$$

DELIMITER ;

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

-- procedure UpdateSpecialOfferAmount

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

DELIMITER $$

USE `Musicervicedb`$$

CREATE DEFINER=`root`@`localhost` PROCEDURE `UpdateSpecialOfferAmount`(IN `specialOfferID` INT, IN `decreaseAmount` INT)

NO SQL

BEGIN

DECLARE newAmount INT;

SET newAmount =

(SELECT Amount FROM specialoffer WHERE SpecialOfferID = specialOfferID) - decreaseAmount;

IF newAmount >= 0

THEN

UPDATE specialoffer

SET Amount = newAmount;

SELECT Amount FROM specialoffer;

END IF;

END$$

DELIMITER ;

USE `Musicervicedb`;

DELIMITER $$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`ValidateEmail\_BeforeInsert`

BEFORE INSERT ON `Musicervicedb`.`user`

FOR EACH ROW

BEGIN

IF (NEW.EmailPromotion != '')

THEN

IF (NEW.EmailPromotion REGEXP '^[A-Z0-9.\_%-]+@[A-Z0-9.-]+.[A-Z]{2,4}$') = 0 THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'EmailPromotion is in incorrect format!';

END IF;

END IF;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`ValidateEmail\_BeforeUpdate`

BEFORE UPDATE ON `Musicervicedb`.`user`

FOR EACH ROW

BEGIN

IF (NEW.EmailPromotion != '')

THEN

IF (NEW.EmailPromotion REGEXP '^[A-Z0-9.\_%-]+@[A-Z0-9.-]+.[A-Z]{2,4}$') = 0 THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'EmailPromotion is in incorrect format!';

END IF;

END IF;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`CheckUserRegistration\_BeforeInsert`

BEFORE INSERT ON `Musicervicedb`.`email\_address`

FOR EACH ROW

BEGIN

SELECT UserID INTO @userID FROM email\_address WHERE UserID = NEW.UserID;

IF @userID IS NOT NULL

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'User is already registered!';

END IF;

IF (NEW.EmailAddress REGEXP '^[A-Z0-9.\_%-]+@[A-Z0-9.-]+.[A-Z]{2,4}$') = 0 THEN

SIGNAL SQLSTATE VALUE '45000'

SET MESSAGE\_TEXT = 'EmailAddress is in incorrect format!';

END IF;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`CheckUserOwnsShoppingCart\_BeforeInsert`

BEFORE INSERT ON `Musicervicedb`.`shoppingcart`

FOR EACH ROW

BEGIN

SELECT UserID INTO @userID FROM shoppingcart WHERE UserID = NEW.UserID;

IF @userID IS NOT NULL

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'User already has a shopping cart';

END IF;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`ValidateScoreValue\_BeforeInsert`

BEFORE INSERT ON `Musicervicedb`.`score`

FOR EACH ROW

BEGIN

IF New.Value < 1 OR New.Value > 10

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Value cannot be less than 1 or more than 10!';

END IF;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`ValidateScoreValue\_BeforeUpdate`

BEFORE UPDATE ON `Musicervicedb`.`score`

FOR EACH ROW

BEGIN

IF New.Value < 1 OR New.Value > 10

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'Value cannot be less than 1 or more than 10!';

END IF;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`CheckPreviousReview\_BeforeInsert`

BEFORE INSERT ON `Musicervicedb`.`review`

FOR EACH ROW

BEGIN

SELECT COUNT(\*) INTO @userReview FROM review WHERE UserID = NEW.UserID AND songID = NEW.songID;

IF @userReview != 0

THEN

SIGNAL SQLSTATE '45000'

SET MESSAGE\_TEXT = 'User has already revied this song! Please, edit your existing review.';

END IF;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`UpdateMusiccore\_AfterDelete`

AFTER DELETE ON `Musicervicedb`.`review`

FOR EACH ROW

BEGIN

SELECT SUM(s.Value) / COUNT(UserID) INTO @totalScore FROM review as R

INNER JOIN Score as S

WHERE R.ScoreID = S.ScoreID AND r.songID = Old.songID;

UPDATE song SET Rating = @totalScore WHERE songID = Old.songID;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`UpdateMusiccore\_AfterInsert`

AFTER INSERT ON `Musicervicedb`.`review`

FOR EACH ROW

BEGIN

SELECT SUM(s.Value) / COUNT(UserID) INTO @totalScore FROM review as R

INNER JOIN Score as S

WHERE R.ScoreID = S.ScoreID AND r.songID = New.songID;

UPDATE song SET Rating = @totalScore WHERE songID = New.songID;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`UpdateMusiccore\_AfterUpdate`

AFTER UPDATE ON `Musicervicedb`.`review`

FOR EACH ROW

BEGIN

SELECT SUM(s.Value) / COUNT(UserID) INTO @totalScore FROM review as R

INNER JOIN Score as S

WHERE R.ScoreID = S.ScoreID AND r.songID = New.songID;

UPDATE song SET Rating = @totalScore WHERE songID = New.songID;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`UpdateShoppingCartTotal\_AfterInsert`

AFTER INSERT ON `Musicervicedb`.`shoppingcart\_item`

FOR EACH ROW

BEGIN

SELECT SUM(Price) INTO @totalPrice FROM shoppingcart\_item

WHERE ShoppingCartID = New.ShoppingCartID;

UPDATE shoppingcart SET TotalPrice = @totalPrice WHERE ShoppingCartID = New.ShoppingCartID;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`DeleteUsersong\_AfterDelete`

AFTER DELETE ON `Musicervicedb`.`user\_subscription`

FOR EACH ROW

BEGIN

DELETE uf.\* FROM user\_song as uf

INNER JOIN subscription\_song as sf

ON uf.songID = sf.songID

WHERE uf.UserID = Old.UserID AND sf.SubscriptionID = Old.SubscriptionID;

END$$

USE `Musicervicedb`$$

CREATE

DEFINER=`root`@`localhost`

TRIGGER `Musicervicedb`.`UpdateUserMusic\_AfterInsert`

AFTER INSERT ON `Musicervicedb`.`user\_subscription`

FOR EACH ROW

BEGIN

INSERT INTO user\_song (UserID, songID, ModifiedDate)

SELECT us.UserID, sf.songID, us.ModifiedDate FROM user\_subscription AS us

INNER JOIN subscription\_song AS sf

ON us.SubscriptionID = sf.SubscriptionID

WHERE us.UserID = New.UserID;

END$$

DELIMITER ;

SET SQL\_MODE=@OLD\_SQL\_MODE;

SET FOREIGN\_KEY\_CHECKS=@OLD\_FOREIGN\_KEY\_CHECKS;

SET [UNIQUE\_CHECKS=@OLD\_UNIQUE\_CHECKS](mailto:UNIQUE_CHECKS%3D@OLD_UNIQUE_CHECKS);

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Обозначение | | | | Наименование | | | | Дополнительные сведения | | | |
|  | | | | Текстовые документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| БГУИР КП 1-40 01 01518  ПЗ | | | | Пояснительная записка | | | | 53 с. | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | | Графические документы | | | |  | | | |
|  | | | |  | | | |  | | | |
| ГУИР 851003 218 ПД | | | | Схема базы данных | | | | Формат А1 | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  | | | |  | | | |  | | | |
|  |  |  |  |  | БГУИР КП 1-40 01 01  518 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм. | Л. | № докум. | Подп. | Дата | «База данных стримингового сервиса для прослушивания музыки»  Ведомость курсового  проекта |  | | | | Лист | Листов |
| Разраб. | | Вайтусенок И.А. |  | 18.12.21 | Т |  | |  | 53 | 53 |
| Пров. | | Марина И.М. |  | 18.12.21 | Кафедра ПОИТ  гр. 851003 | | | | | |
|  | |  |  |  |