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](#_Toc59157864)

[1 Анализ предметной области 5](#_Toc59157865)

[1.1 Обзор аналогов 5](#_Toc59157866)

[1.2 Постановка задачи 7](#_Toc59157867)

[2 Разработка базы данных 9](#_Toc59157868)

[2.1 Выбор СУБД и средств разработки 9](#_Toc59157869)

[2.1.1. Типы баз данных 9](#_Toc59157870)

[2.1.2. Язык SQL 10](#_Toc59157871)

[2.1.3. Выбор СУБД 10](#_Toc59157872)

[2.1.4. Выбор средств разработки 10](#_Toc59157873)

[2.2 Особенности реализации 11](#_Toc59157874)

[2.3 Пользователи системы и их роли 12](#_Toc59157875)

[3 Модель предметной области 13](#_Toc59157876)

[3.1 Понятие инфологической модели 13](#_Toc59157877)

[3.2 Выделение объектов предметной области 13](#_Toc59157878)

[3.3 Выделение атрибутов объектов предметной области 13](#_Toc59157879)

[3.4 Бизнес-логика 24](#_Toc59157880)

[3.5 Проверка на нормализацию 26](#_Toc59157881)

[4 Тестирование базы данных 27](#_Toc59157882)

[ЗАКЛЮЧЕНИЕ 28](#_Toc59157883)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29](#_Toc59157884)

[Приложение А Схема базы данных на языке SQL 30](#_Toc59157885)

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, который сможет предоставлять пользователям возможность сохранять информацию о кино, получать о них необходимую информацию, а также высказывать свое мнение и выставлять оценки фильмам и сериалам.

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

Сервис IMDb, или Internet Movie Database, является крупнейшей в мире базой данных и сайтом о кинематографе. В базе собрана информация о более чем 6,5 миллионах кинофильмов, телесериалов и отдельных серий, а также о 10,4 миллионах персоналий, связанных с кино, — актерах, режиссерах, сценаристах и др. Одной из главных особенностей IMDb является то, что редактировать информацию в базе данных может любой пользователь, прошедший регистрацию и модерацию на сайте. Пример страницы фильма на сайте IMDb отображена на рисунке 1.1.

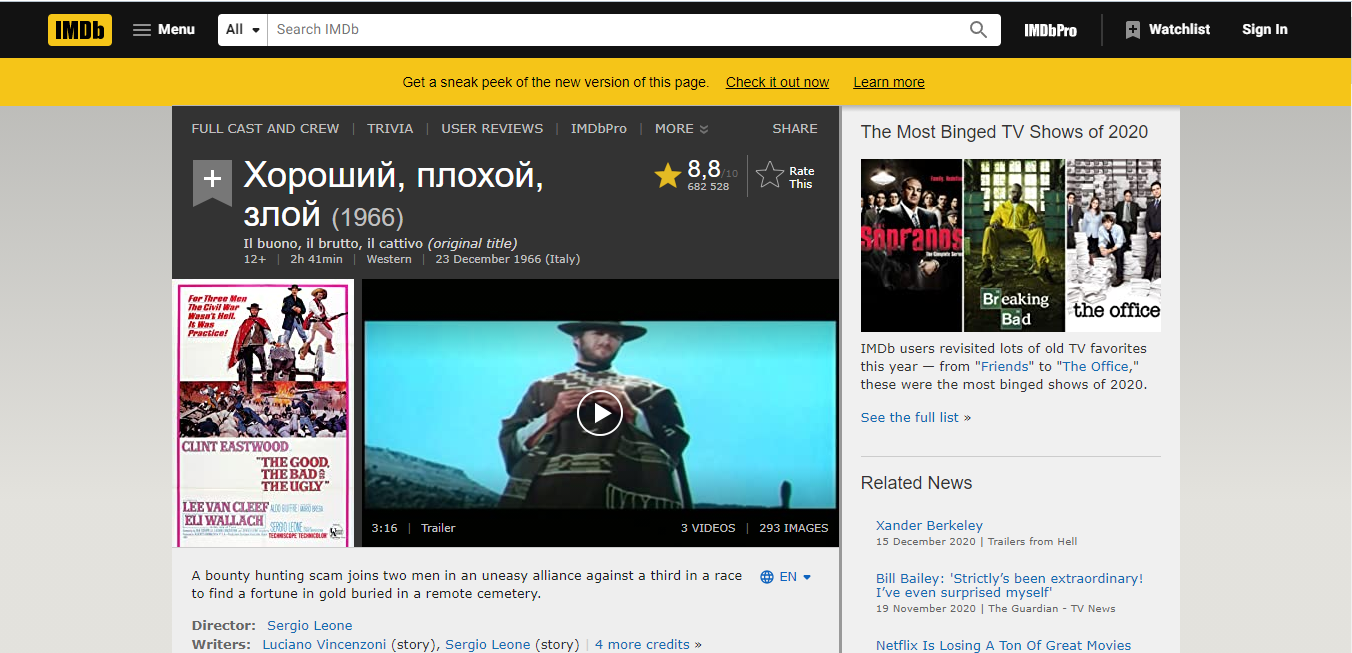


Рисунок 1.1 – страница фильма на сайте IMDb

На сайте содержится много полезной информации о фильме, например, его продолжительность, возрастные ограничения, видео- и фотоматериалы, актерский состав. Каждый пользователь может написать рецензию к фильму и поставить свою оценку фильму, из которой потом формируется средний рейтинг фильма. По этому рейтингу в дальнейшем создается список из наиболее хорошо оцененных фильмов.

Сайт позволяет формировать список фильмов, который пользователь хочет посмотреть, а также создавать свои списки. Также на сайте можно просмотреть список всех сезонов и серий для каждого сериала, а также краткое описание к ним, что является очень полезной функцией.

В качестве минуса можно только отметить наличие на данном сайте помимо кино информации о компьютерных играх, хотя сайт по большей части специализируется именно на кино.

Следующим аналогом является сервис КиноПоиск. Данный сайт очень похож по функциональности на IMDb, но является более популярным в русскоговорящих странах и считается крупнейшим русскоязычным сервисом о кино.

Из особенностей данного сайта можно подчеркнуть наличие онлайн-кинотеатра, что позволяет просматривать фильмы сразу на сайте по подписке.

Страница фильма на данном сайте изображена на рисунке 1.2.

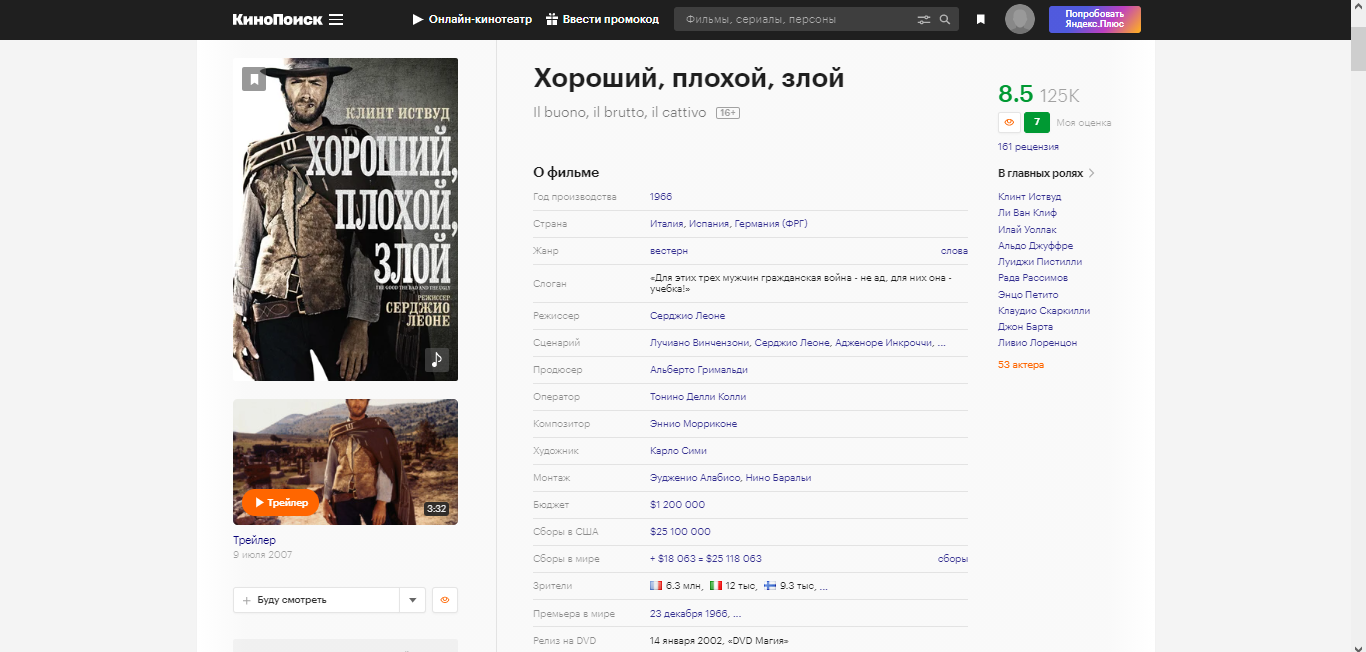


Рисунок 1.2 – пример страницы фильма на сервисе КиноПоиск

Данный сервис позволяет пользователям выставлять оценки фильмам, писать рецензии, а также сортировать фильмы по папкам. Еще в качестве плюса можно отметить наличие статистики для каждого пользователя, которая позволяет анализировать свои предпочтения, а также смотреть людей, с наиболее схожими интересами.

Еще одним из положительных качеств является наличие на сайте информации о кинотеатрах: их местоположение, время работы, сайты, номера телефонов.

В качестве недостатком сайта можно отметить отсутствие подробной информации о каждой серии или сезоне сериала, из-за чего приходится искать такую информацию на других ресурсах.

Последним приведенным аналогом является раздел Кино сайта Афиша TUT.BY. Данный сайт направлен больше на получение информации о киносеансах и обладает не очень большим объемом информации, а также направлен больше на белорусскую аудиторию. Пример страницы фильма показан на рисунке 1.3.

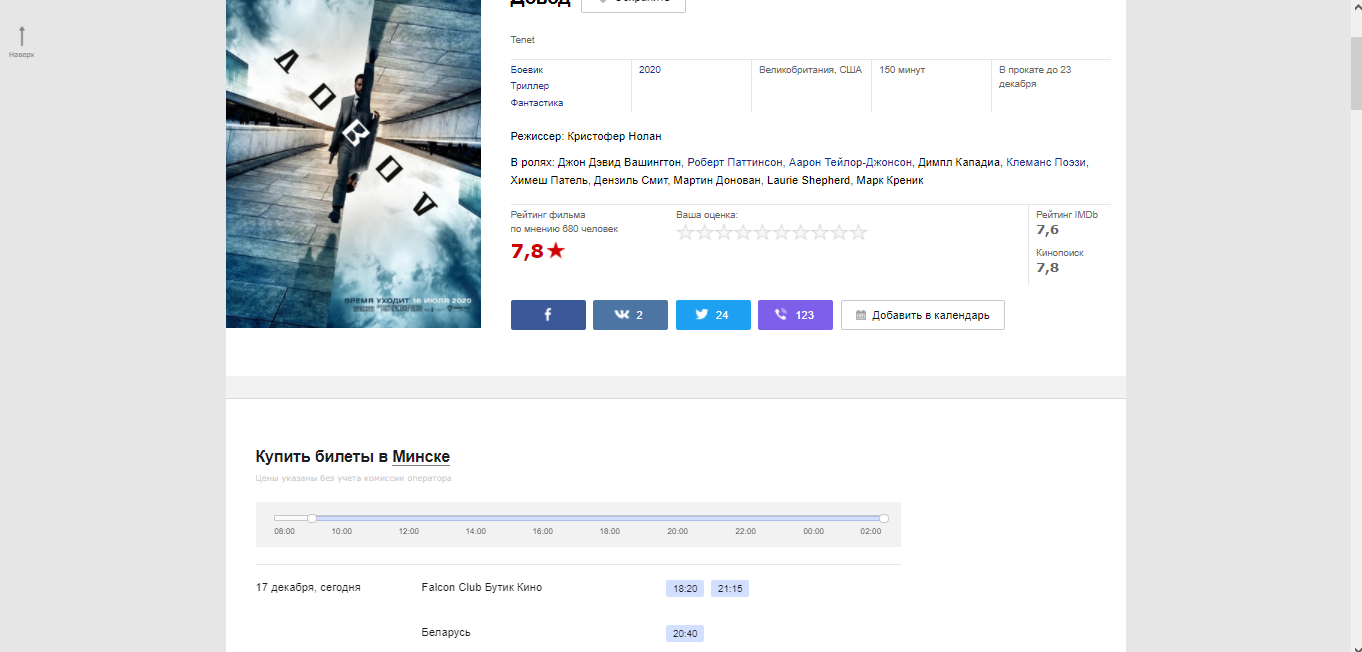


Рисунок 1.3 – пример страницы фильма на сервисе Афиша TUT.BY

Данный сайт содержит только фильмы, которые выходят в прокате. Тут также можно выставлять оценки фильмам и писать рецензии. Несмотря на скудность данных по сравнению с предыдущими аналогами, сервис очень удобен для просмотра киносеансов и покупки билетов в кинотеатры в Беларуси.

* 1. Постановка задачи

В рамках данного курсового проекта планируется спроектировать базу данных, которую в дальнейшем можно будет использовать в различных киносервисах.

При проектировании базы данных планируется выполнить следующие задачи:

– регистрация пользователя в сервисе;

– возможность создания пользователем папок с фильмами;

– возможность пользователю выставлять оценки фильму и оставлять комментарии;

– создание различных хранимых процедур и триггеров для удобной работы с данными;

– создание индексов для ускорения поиска по данным сервиса;

– нормализовать полученную базу данных.

В рамках курсового проекта будет спроектирована и разработана реляционная база данных, содержащая сущности и связи, описывающие предметную область киносервиса (фильм, награда, фестиваль, киностудия и т.д.).

Для проектирования базы данных планируется использовать реляционную систему управления базами данных MySQL, а также инструмент для визуального проектирования баз данных MySQL Workbench.

1. Разработка базы данных
   1. Выбор СУБД и средств разработки
      1. Типы баз данных

Существует множество различных типов баз данных. Необходимый тип выбирается индивидуально для каждой предметной области, в зависимости от ожидаемого функционала и возможностей разрабатываемого приложения.

Некоторые типы баз данных:

– картотеки;

– сетевые базы данных;

– иерархические базы данных;

– объектно-ориентированные базы данных;

– реляционные базы данных;

– дедуктивные базы данных;

– NoSQL базы данных.

Каждый из типов имеет свои преимущества и недостатки.

Наиболее подходящими могут быть следующие типы: картотека, реляционная, объектно-ориентированная или NoSQL базы данных.

Картотека представляет собой упорядоченное (по алфавиту, дате и т.п.) собрание данных в виде записей, каждая из которых предоставляет сведения о каком-то объекте базы данных.

Реляционные базы данных основаны на теоретико-множественной реляционной даталогической модели. Все данные представлены в виде связанных между собой таблиц, разбитых на строки и столбцы.

В объектно-ориентированных базах данных данные оформлены в виде моделей объектов, включающих прикладные программы, которые управляются внешними событиями. Это технология напоминает объектно-ориентированное программирование в применении к БД.

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.

В рамках данного курсового проекта была выбрана реляционная модель базы данных, так как мы имеем структурированные данные. Реляционная модель понятна и легко визуализируема. Существует множество инструментов для удобной работы с реляционными базами данных. Теоретическая база для данного вида баз данных разрабатывалась долгое время и имеет конкретное математическое обоснование своей эффективности.

* + 1. Язык SQL

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

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, но 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, так как она является свободной и удобной реляционной СУБД. Ее разработку и поддержку которой осуществляет корпорация Oracle.

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.

* + 1. Выбор средств разработки

В качестве средства разработки базы данных был выбран инструмент для визуального проектирования баз данных MySQL Workbench, интегрирующий проектирование, моделирование, создание и эксплуатацию базы данных в единое бесшовное окружение для системы баз данных MySQL. Он очень удобен для проектирования больших баз данных, так как предоставляет визуальный интерфейс для разработки. Пример создания таблиц базы данных в MySQL Workbench представлен на рисунке 2.1.

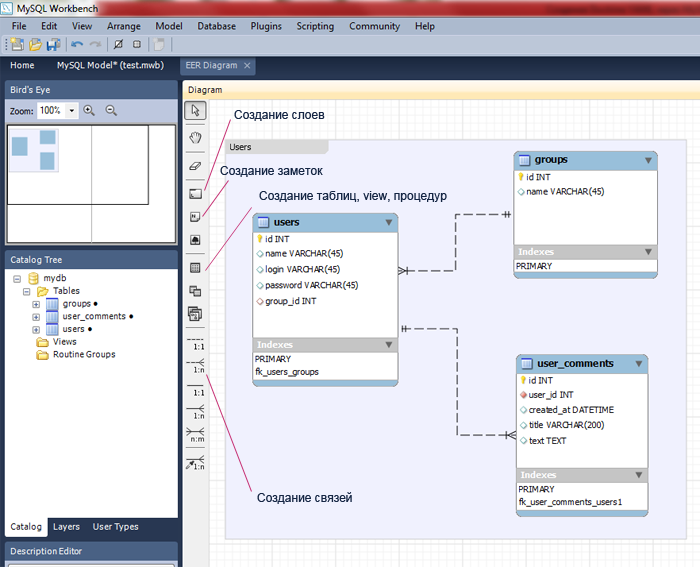


Рисунок 2.1 – пример создания таблиц базы данных в MySQL Workbench

Ниже приведены некоторые преимущества инструмента MySQL Workbench:

– позволяет наглядно представить модель базы данных в графическом виде;

– наглядный и функциональный механизм установки связей между таблицами;

– удобный редактор SQL запросов, позволяющий сразу же отправлять их серверу и получать ответы в виде таблицы;

– возможность редактирования данных в таблице в визуальном режиме.

* 1. Особенности реализации

Далее перечислен основной функционал, реализованный в базе данных:

– возможность добавления фильма пользователем в одну из его папок;

– возможность выставления оценки фильму;

– вычисление среднего рейтинга фильма;

– выставление эмоции like/dislike статьям, франшизам, странам и жанрам, чтобы в дальнейшем можно было выстраивать рекомендательные системы с помощью существующей базы данных;

– возможность создания ассоциации между статьей и фильмом, персоной, студией;

– возможность создания ассоциации между фото и фильмом, персоной, студией.

Также было добавлено использования индексов для ускорения поиска по названию фильма, фамилии знаменитости, кинотеатра по названию, статьи по заглавию и т.д.

При разработке схемы базы данных было уделено внимание безопасности. Аккаунт пользователя защищен паролем, при этом в базе данных хранится только его хэш.

* 1. Пользователи системы и их роли

Для сервиса о кино было принято выделить несколько основных типов пользователей.

Первым типом пользователей является гость. Это пользователи, зашедшие на сервис и не прошедшие процедуры авторизации. Такие пользователи могут только просматривать существующие фильмы, знаменитостей, награды, фестивали и т.д. Также они могут оставлять анонимные комментарии под фильмом.

Второй тип пользователей – это юзеры. Они являются пользователями, которые когда-то выполнили регистрацию и авторизовались на сайте. Для таких пользователей функционал намного шире, чем для гостей. Они могут создавать свои папки, ставить оценки фильмам, выбирать понравившиеся статьи, жанры, страны. Также они могут отмечать, на какие сеансы они идут. При разработке приложения это позволит добавить функционал напоминания о ближайших киносеансах.

Следующий тип пользователей – модераторы. Они могут не только просматривать информацию, но и редактировать ее. Это позволит сделать информацию наиболее достоверной и понятной.

Последним типом является администратор. Такие пользователи имеют функционал модераторов, при этом они могут сами назначать и удалять модераторов.

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:

– видео;

– жанр;

– комментарий;

– киносеанс;

– кинотеатр;

– награда;

– персона;

– пользователь;

– роль пользователя;

– страна;

– студия;

– сериал;

– сезон;

– саундтрек;

– статья;

– фильм;

– фестиваль;

– фото;

– франшиза;

– эмоция;

– эпизод.

* 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) | idUser | int | идентификатор | первичный ключ |
| login | varchar(255) | имя пользователя | не пустой, уникальный |
| password\_hash | varchar(255) | хэш пароля | не пустой |
| mail | varchar(255) | электронная почта | уникальный |
| User\_Role\_id | int | ид. роли пользователя | не пустой, внешний ключ |
| 2 | User\_Role (роль пользователя) | idUser\_Role | int | идентификатор | первичный ключ |
| name | varchar(9) | название | не пустой, уникальный |
| 3 | Article (статья) | idArticle | int | идентификатор | первичный ключ |
| title | varchar(255) | заголовок | не пустой |
| description | varchar(255) | описание |  |
| text\_location | varchar(255) | расположение текста на сервере | не пустой |
| date | date | дата |  |
| 4 | Video (видео) | idVideo | int | идентификатор | первичный ключ |
| title | varchar(255) | название | не пустой |
| description | varchar(255) | описание |  |
| video\_location | varchar(255) | расположение видео на сервере | не пустой |
| 5 | Photo (фото) | idPhoto | Int | идентификатор | первичный ключ |
| title | varchar(255) | название | не пустой |
| description | varchar(255) | описание |  |
| photo\_location | varchar(255) | расположение фото на сервере | не пустой |
| 6 | Film (фильм) | idFilm | int | идентификатор | первичный ключ |
| release\_date | date | дата выхода |  |
| name | varchar(255) | название |  |
| duration | int | продолжительность (в минутах) |  |
| age\_restriction | int | возрастные ограничения |  |
| 7 | Genre (жанр) | IdGenre | int | идентификатор | первичный ключ |
| name | varchar(45) | название | не пустой, уникальный |
| 8 | Comment (комментарий) | idComment | int | идентификатор | первичный ключ |
| Film\_id | int | ид. фильма | внешний ключ, не пустой |
| User\_id | int | ид. пользователя | внешний ключ |
| text | varchar(255) | текст | не пустой |
| 9 | Country (страна) | idCountry | int | идентификатор | первичный ключ |
| name | varchar(255) | название | не пустой, уникальный |
| 10 | Emotion (эмоция) | like | bit | идентификатор (0 – не нравится, 1 - нравится) | первичный ключ |
| 11 | Studio (студия) | idStudio | int | идентификатор | первичный ключ |
| name | varchar(255) | название | не пустой |
| description | varchar(255) | описание |  |
| foundation\_date | date | дата основания |  |
| 12 | Person  (личность) | idPerson | int | идентификатор | первичный ключ |
| birth\_date | date | дата рождения |  |
| death\_date | date | дата смерти |  |
| first\_name | varchar(255) | имя |  |
| second\_name | varchar(255) | фамилия |  |
| description | varchar(255) | описание |  |
| 13 | Award (награда) | idAward | int | идентификатор | первичный ключ |
| name | varchar(255) | название | не пустой |
| description | varchar(255) | описание |  |
| location | varchar(255) | местоположение |  |
| date | date | дата |  |
| 14 | Festival (фестиваль) | idFestival | int | идентификатор | первичный ключ |
| name | varchar(255) | название | не пустой |
| date | date | дата |  |
| description | varchar(255) | описание |  |
| location | varchar(255) | местоположение |  |
| 15 | Franchise (франшиза) | idFranchise | int | идентификатор | первичный ключ |
| name | varchar(255) | название | не пустой |
| description | varchar(255) | описание |  |
| 16 | Cinema (кинотеатр) | idCinema | int | идентификатор | первичный ключ |
| name | varchar(255) | название | не пустой |
| address | varchar(255) | адрес | не пустой |
| phone\_number | varchar(255) | номер телефона |  |
| description | varchar(255) | описание |  |
| website | varchar(255) | сайт |  |
| 17 | Soundtrack (суандтрек) | idSoundtrack | int | идентификатор | первичный ключ |
| Film\_id | int | ид. фильма | внешний ключ, не пустой |
| soundtrack\_location | varchar(255) | местоположение на сервере |  |
| description | varchar(255) | Описание |  |
| time\_in\_film | int | время в фильме (в минутах) |  |
| 18 | Series (сериал) | idSeries | int | идентификатор | первичный ключ |
| Film\_id | int | ид. пользователя, на которого подписан данный пользователь | внешний ключ, не пустой |
| closing\_date | date | дата закрытия |  |
| 19 | Season (сезон) | idSeason | int | идентификатор | первичный ключ |
| Series\_id | int | ид. сериала | внешний ключ, не пустой |
| name | varchar(255) | название |  |
| number | int | порядковый номер | не пустой |
| start\_date | date | дата начала |  |
| end\_date | date | дата окончания |  |
| description | varchar(255) | описание |  |
| 20 | Episode (эпизод) | idEpisode | int | идентификатор | первичный ключ |
| Season\_id | int | ид. сезона | внешний ключ, не пустой |
| name | varchar(255) | название |  |
| number | int | порядковый номер | не пустой |
| description | varchar(255) | описание |  |
| duration | Int | продолжительность в минутах |  |
| 21 | User\_Article  (статья юзера) | User\_id | int | ид. пользователя | первичный ключ, внешний ключ |
| Article\_id | int | ид. статьи | первичный ключ, внешний ключ |
| Emotion\_like | bit | эмоция | не пустой |
| 22 | User\_Genre (жанр пользователя) | Genre\_id | int | ид. жанра | первичный ключ, внешний ключ |
| User\_id | int | ид. пользователя | первичный ключ, внешний ключ |
| Emotion\_like | bit | эмоция | не пустой |
| 23 | Film\_Genre (жанр фильма) | Film\_id | int | ид. Фильма | первичный ключ, внешний ключ |
| Genre\_id | int | ид. жанра | первичный ключ, внешний ключ |
| 24 | User\_Film (фильм пользователя) | User\_id | int | ид. пользователя | первичный ключ, внешний ключ |
| Film\_id | int | ид. фильма | первичный ключ, внешний ключ |
| User\_folder\_id | int | ид. папки пользователя | первичный ключ, внешний ключ |
| score | int | Оценка |  |
| 25 | Film\_Country (страна фильма) | Country\_id | int | ид. Страны | первичный ключ, внешний ключ |
| Film\_id | int | ид. фильма | первичный ключ, внешний ключ |
| 26 | Studio\_Film (фильм студии) | Studio\_id | int | ид. Студии | первичный ключ, внешний ключ |
| Film\_id | int | ид. фильма | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 27 | User\_Folder (папка пользователя) | idUser\_Folder | int | идентификатор | первичный ключ |
| name | varchar(255) | название | не пустой |
| User\_id | int | ид. пользователя | не пустой, внешний ключ |
| 28 | Person\_in\_Film (личность в фильме) | IdPerson\_in\_Film | int | идентификатор | первичный ключ |
| Person\_id | int | ид. Личности | внешний ключ, не пустой |
| Film\_id | int | ид. фильма | внешний ключ, не пустой |
| description | varchar(255) | описание |  |
| role | varchar(255) | роль |  |
| 29 | Film\_in\_Article (фильм в статье) | Article\_id | int | ид. Статьи | первичный ключ, внешний ключ |
| Film\_id | int | ид. фильма | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 30 | User\_Franchise (франшиза пользователя) | Franchise\_id | int | ид. франшизы | первичный ключ, внешний ключ |
| User\_id | int | ид. пользователя | первичный ключ, внешний ключ |
| Emotion\_like | bit | эмоция | не пустой |
| 31 | Studio\_Country\_office (офис студии в стране) | idStudio\_Country\_office | int | идентификатор | первичный ключ |
| Studio\_id | int | ид. студии | внешний ключ, не пустой |
| Country\_id | int | ид. страны | внешний ключ, не пустой |
| website | varchar(255) | сайт |  |
| address | varchar(255) | адрес |  |
| foundation\_date | date | дата основания |  |
| description | varchar(255) | описание |  |
| 32 | User\_country (страна пользователя) | Country\_id | int | ид. страны | первичный ключ, внешний ключ |
| User\_id | int | ид. пользователя | первичный ключ, внешний ключ |
| Emotion\_like | bit | эмоция | не пустой |
| 33 | Film\_video (видео фильма) | Video\_id | int | ид. Видео | первичный ключ, внешний ключ |
| Film\_id | Int | ид. фильма | первичный ключ, внешний ключ |
| name | varchar(255) | название |  |
| description | varchar(255) | описание |  |
| 34 | Person\_on\_Festival (личность на фестивале) | Festival\_id | int | ид. фестиваля | первичный ключ, внешний ключ |
| Person\_id | int | ид. личности | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 35 | Festival\_Award (награда фестиваля) | idFestival\_Award | int | идентификатор | первичный ключ |
| Festival\_id | int | ид. фестиваля | внешний ключ, не пустой |
| Award\_id | int | ид. награды | внешний ключ, не пустой |
| description | varchar(255) | описание |  |
| 36 | Person\_Award (награда личности) | idPerson\_Award | int | идентификатор | первичный ключ |
| Person\_id | Int | ид. личности | внешний ключ, не пустой |
| Award\_id | Int | ид. награды | внешний ключ, не пустой |
| description | varchar(255) | описание |  |
| is\_winner | bit | победитель |  |
| 37 | Person\_in\_Article (личность в статье) | Article\_id | int | ид. статьи | первичный ключ, внешний ключ |
| Person\_id | int | ид. личности | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 38 | User\_Film\_session  (киносеанс пользователя) | Film\_Session\_id | int | ид. киносеанса | первичный ключ, внешний ключ |
| User\_id | int | ид. пользователя | первичный ключ, внешний ключ |
| 39 | Studio\_in\_Article (студия в статье) | Article\_id | int | ид. Статьи | первичный ключ, внешний ключ |
| Studio\_id | int | ид. студии | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 40 | Studio\_Award (награда студии) | idStudio\_Award | int | идентификатор | первичный ключ |
| Studio\_id | int | ид. студии | внешний ключ, не пустой |
| Award\_id | int | ид. награды | внешний ключ, не пустой |
| description | varchar(255) | описание |  |
| is\_winner | bit | победитель |  |
| 41 | Film\_Award (награда фильма) | idFilm\_Award | int | идентификатор | первичный ключ |
| Film\_id | Int | ид. фильма | внешний ключ, не пустой |
| Award\_id | Int | ид. награды | внешний ключ, не пустой |
| description | varchar(255) | описание |  |
| is\_winner | bit | победитель |  |
| 42 | Film\_Festival (фильм на фестивале) | idFilm\_Festival | int | идентификатор | первичный ключ |
| Festival\_id | int | ид. фестиваля | внешний ключ, не пустой |
| Film\_id | int | ид. фильма | внешний ключ, не пустой |
| description | varchar(255) | описание |  |
| 43 | Film\_in\_franchise (фильм в франшизе) | idFilm\_in\_Franchise | int | идентификатор | первичный ключ |
| Franchise\_id | int | ид. франшизы | внешний ключ, не пустой |
| Film\_id | int | ид. фильма | внешний ключ, не пустой |
| description | varchar(255) | описание |  |
| number | int | порядковый номер | не пустой |
| 44 | Photo\_in\_Article (фото в статье) | Article\_id | Int | ид. статьи | первичный ключ, внешний ключ |
| Photo\_id | Int | ид. фото | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 45 | Studio\_Photo (фото студии) | Studio\_id | int | ид. студии | первичный ключ, внешний ключ |
| Photo\_id | int | ид. фото | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 46 | Video\_in\_Article (видео в статье) | Article\_id | int | ид. статьи | первичный ключ, внешний ключ |
| Video\_id | int | ид. видео | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 47 | Person\_Photo (фото личности) | Person\_id | int | ид. личности | первичный ключ, внешний ключ |
| Photo\_id | Int | ид. фото | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 48 | Film\_Photo (фото фильма) | Film\_id | int | ид. фильма | первичный ключ, внешний ключ |
| Photo\_id | int | ид. фото | первичный ключ, внешний ключ |
| description | varchar(255) | описание |  |
| 49 | Film\_session (киносеанс) | idFilm\_Session | int | идентификатор | первичный ключ |
| Cinema\_id | int | ид. кинотеатра | внешний ключ, не пустой |
| Film\_id | int | ид. фильма | внешний ключ, не пустой |
| screening\_date | date | дата показа | не пустой |
| description | varchar(255) | описание |  |

Таблица 3.1 – Представление сущностей и атрибутов базы данных

* 1. Бизнес-логика

Бизнес-логика в базе данных представлена триггерами, процедурами и индексами.

Так как база данных специализируется на хранении информации о фильмах, все пользователи будут искать интересующие их фильмы именно по названию. Для того, чтобы ускорить процедуру поиска, были построены индексы по названию фильмов. Код создания такого индекса приведен ниже.

CREATE INDEX `find\_by\_name` ON `film\_service\_db`.`Film` (`name` ASC) VISIBLE;

Также индексы построены по поиску личности по имени или фамилии. Так как названия жанров и стран уникальны, то для них поставлены уникальные индексы.

CREATE UNIQUE INDEX `name\_UNIQUE` ON `film\_service\_db`.`Country` (`name` ASC) VISIBLE;

CREATE UNIQUE INDEX `name\_UNIQUE` ON `film\_service\_db`.`Genre` (`name` ASC) VISIBLE;

Поскольку в информации о звездных личностях хранятся даты рождения и смерти, они не могут превышать текущую дату. Поэтому было принято решение написать функцию и выставить триггеры, которые при выставлении даты, превышающей текущую, возвращают предыдущее значение. Код такой функции и триггера отображены далее.

CREATE DEFINER = CURRENT\_USER TRIGGER

film\_service\_db.Person\_BEFORE\_UPDATE

BEFORE UPDATE ON Person

FOR EACH ROW

BEGIN

IF NEW.birth\_date > DATE(NOW()) THEN

SET NEW.birth\_date =

get\_old\_birth\_date(NEW.idPerson);

END IF;

IF NEW.death\_date > DATE(NOW()) THEN

SET NEW.death\_date =

get\_old\_death\_date(NEW.idPerson);

END IF;

END$$

USE `film\_service\_db`$$

CREATE FUNCTION `get\_old\_birth\_date` (person\_id INT)

RETURNS DATE DETERMINISTIC

BEGIN

RETURN (SELECT birth\_date FROM Person

WHERE id = person\_id ORDER BY id DESC LIMIT 1);

END$$

* 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

В данной главе приведены примеры тестирования спроектированной базы данных.

Для тестирования разработанной базы данных будут использоваться запросы, приведенные ниже.

Добавление записи в таблицу. Код добавления нового фильма приведен на рисунке ниже.

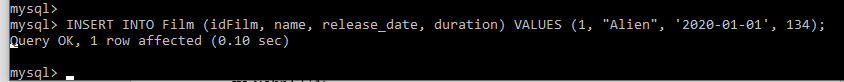


Рисунок 4.1 – пример добавления фильма в таблицу Film

Видим, что строка успешно добавлена.

Далее обновим добавленную запись. Код обновления представлен ниже.

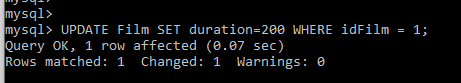


Рисунок 4.2 – пример обновления фильма в таблицу Film

Удалим теперь добавленную запись. Код удаления представлен ниже.

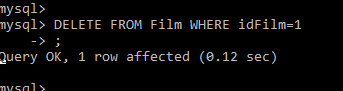


Рисунок 4.3 – пример удаления фильма в таблицу Film

Запись успешно удалена.

В ходе запуска скриптов базы данных и их тестирования ошибок не было обнаружено.

ЗАКЛЮЧЕНИЕ

Кино в наши дни является одним из самых популярных и прибыльных видов искусства. В рамках данного курсового проекта была разработана база данных, которую можно в дальнейшем использовать для создания информационного сервиса о кино. Для разработки использовался инструмент MySQL Workbench и Java и СУБД MySQL.

В процессе разработки были получены навыки проектирования базы данных и работы с СУБД MySQL.

Полученная база данных приведена к третьей нормальной форме, для работы с данными были созданы функции и триггеры, построены индексы.

Итоговая база данных детально покрывает предметную область и позволяет на ее основе строить различные сервисы о кино.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. <https://docs.spring.io/spring/docs/current/spring-framework-reference/index.html>
2. Справка MySQL 8.0 [электронный ресурс]. – Электронные данные. – Режим доступа: <https://dev.mysql.com/doc/refman/8.0/en/>
3. SQL – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/SQL
4. MySQL – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/MySQL
5. Реляционная СУБД – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/Реляционная СУБД
6. Нормальная форма – Электронные данные. – Режим доступа: <https://ru.wikipedia.org/>wiki/Нормальная форма

Приложение А  
Схема 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 film\_service\_db

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

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

-- Schema film\_service\_db

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

CREATE SCHEMA IF NOT EXISTS `film\_service\_db` DEFAULT CHARACTER SET utf8 ;

USE `film\_service\_db` ;

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

-- Table `film\_service\_db`.`User\_Role`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Role` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Role` (

`idUser\_Role` INT NOT NULL,

`name` VARCHAR(9) NOT NULL,

PRIMARY KEY (`idUser\_Role`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `name\_UNIQUE` ON `film\_service\_db`.`User\_Role` (`name` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`User`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User` (

`idUser` INT NOT NULL,

`User\_Role\_id` INT NOT NULL,

`login` VARCHAR(255) NOT NULL,

`password\_hash` VARCHAR(255) NOT NULL,

`mail` VARCHAR(255) NULL,

PRIMARY KEY (`idUser`),

CONSTRAINT `fk\_User\_User\_Role1`

FOREIGN KEY (`User\_Role\_id`)

REFERENCES `film\_service\_db`.`User\_Role` (`idUser\_Role`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_User\_Role1\_idx` ON `film\_service\_db`.`User` (`User\_Role\_id` ASC) VISIBLE;

CREATE UNIQUE INDEX `mail\_UNIQUE` ON `film\_service\_db`.`User` (`mail` ASC) VISIBLE;

CREATE UNIQUE INDEX `login\_UNIQUE` ON `film\_service\_db`.`User` (`login` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Film`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film` (

`idFilm` INT NOT NULL,

`name` VARCHAR(255) NULL,

`release\_date` DATE NULL,

`duration` INT NULL,

`age\_restriction` INT NULL,

PRIMARY KEY (`idFilm`))

ENGINE = InnoDB;

CREATE INDEX `find\_by\_name` ON `film\_service\_db`.`Film` (`name` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`User\_Folder`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Folder` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Folder` (

`idUser\_Folder` INT NOT NULL,

`name` VARCHAR(255) NOT NULL,

`User\_id` INT NOT NULL,

PRIMARY KEY (`idUser\_Folder`),

CONSTRAINT `fk\_User\_Folder\_User1`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_Folder\_User1\_idx` ON `film\_service\_db`.`User\_Folder` (`User\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`User\_Film`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Film` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Film` (

`User\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`User\_Folder\_id` INT NOT NULL,

`score` INT NULL,

PRIMARY KEY (`User\_Folder\_id`, `Film\_id`, `User\_id`),

CONSTRAINT `fk\_User\_Film\_User`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Film\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Film\_User\_Folder1`

FOREIGN KEY (`User\_Folder\_id`)

REFERENCES `film\_service\_db`.`User\_Folder` (`idUser\_Folder`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_Film\_Film1\_idx` ON `film\_service\_db`.`User\_Film` (`Film\_id` ASC) VISIBLE;

CREATE INDEX `fk\_User\_Film\_User\_Folder1\_idx` ON `film\_service\_db`.`User\_Film` (`User\_Folder\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Studio`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Studio` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Studio` (

`idStudio` INT NOT NULL,

`name` VARCHAR(255) NOT NULL,

`description` VARCHAR(255) NULL,

`foundation\_date` DATE NULL,

PRIMARY KEY (`idStudio`))

ENGINE = InnoDB;

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

-- Table `film\_service\_db`.`Studio\_Film`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Studio\_Film` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Studio\_Film` (

`Studio\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Studio\_id`, `Film\_id`),

CONSTRAINT `fk\_Studio\_Film\_Studio1`

FOREIGN KEY (`Studio\_id`)

REFERENCES `film\_service\_db`.`Studio` (`idStudio`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Studio\_Film\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Studio\_Film\_Film1\_idx` ON `film\_service\_db`.`Studio\_Film` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Country`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Country` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Country` (

`idCountry` INT NOT NULL,

`name` VARCHAR(255) NOT NULL,

PRIMARY KEY (`idCountry`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `name\_UNIQUE` ON `film\_service\_db`.`Country` (`name` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Studio\_Country\_office`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Studio\_Country\_office` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Studio\_Country\_office` (

`idStudio\_Country\_office` INT NOT NULL,

`Studio\_id` INT NOT NULL,

`Country\_id` INT NOT NULL,

`website` VARCHAR(255) NULL,

`address` VARCHAR(255) NULL,

`foundation\_date` DATE NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idStudio\_Country\_office`),

CONSTRAINT `fk\_Studio\_Country\_office\_Studio1`

FOREIGN KEY (`Studio\_id`)

REFERENCES `film\_service\_db`.`Studio` (`idStudio`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Studio\_Country\_office\_Country1`

FOREIGN KEY (`Country\_id`)

REFERENCES `film\_service\_db`.`Country` (`idCountry`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Studio\_Country\_office\_Studio1\_idx` ON `film\_service\_db`.`Studio\_Country\_office` (`Studio\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Studio\_Country\_office\_Country1\_idx` ON `film\_service\_db`.`Studio\_Country\_office` (`Country\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Film\_Country`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_Country` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_Country` (

`Country\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

PRIMARY KEY (`Country\_id`, `Film\_id`),

CONSTRAINT `fk\_Film\_Country\_Country1`

FOREIGN KEY (`Country\_id`)

REFERENCES `film\_service\_db`.`Country` (`idCountry`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_Country\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_Country\_Film1\_idx` ON `film\_service\_db`.`Film\_Country` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Genre`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Genre` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Genre` (

`idGenre` INT NOT NULL,

`name` VARCHAR(45) NOT NULL,

PRIMARY KEY (`idGenre`))

ENGINE = InnoDB;

CREATE UNIQUE INDEX `name\_UNIQUE` ON `film\_service\_db`.`Genre` (`name` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Film\_Genre`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_Genre` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_Genre` (

`Film\_id` INT NOT NULL,

`Genre\_id` INT NOT NULL,

PRIMARY KEY (`Film\_id`, `Genre\_id`),

CONSTRAINT `fk\_Film\_Genre\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_Genre\_Genre1`

FOREIGN KEY (`Genre\_id`)

REFERENCES `film\_service\_db`.`Genre` (`idGenre`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_Genre\_Genre1\_idx` ON `film\_service\_db`.`Film\_Genre` (`Genre\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Person`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Person` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Person` (

`idPerson` INT NOT NULL,

`birth\_date` DATE NULL,

`death\_date` DATE NULL,

`first\_name` VARCHAR(255) NULL,

`second\_name` VARCHAR(255) NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idPerson`))

ENGINE = InnoDB;

CREATE INDEX `name\_family` ON `film\_service\_db`.`Person` (`first\_name` ASC, `second\_name` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Person\_in\_Film`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Person\_in\_Film` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Person\_in\_Film` (

`idPerson\_in\_Film` INT NOT NULL,

`Person\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`role` VARCHAR(255) NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idPerson\_in\_Film`),

CONSTRAINT `fk\_Person\_in\_Film\_Person1`

FOREIGN KEY (`Person\_id`)

REFERENCES `film\_service\_db`.`Person` (`idPerson`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Person\_in\_Film\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Person\_in\_Film\_Person1\_idx` ON `film\_service\_db`.`Person\_in\_Film` (`Person\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Person\_in\_Film\_Film1\_idx` ON `film\_service\_db`.`Person\_in\_Film` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Award`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Award` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Award` (

`idAward` INT NOT NULL,

`name` VARCHAR(255) NOT NULL,

`description` VARCHAR(255) NULL,

`location` VARCHAR(255) NULL,

`date` DATE NULL,

PRIMARY KEY (`idAward`))

ENGINE = InnoDB;

CREATE INDEX `name` ON `film\_service\_db`.`Award` (`name` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Film\_Award`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_Award` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_Award` (

`idFilm\_Award` INT NOT NULL,

`Award\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

`is\_winner` BIT NULL,

PRIMARY KEY (`idFilm\_Award`),

CONSTRAINT `fk\_Film\_Award\_Award1`

FOREIGN KEY (`Award\_id`)

REFERENCES `film\_service\_db`.`Award` (`idAward`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_Award\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_Award\_Award1\_idx` ON `film\_service\_db`.`Film\_Award` (`Award\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Film\_Award\_Film1\_idx` ON `film\_service\_db`.`Film\_Award` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Studio\_Award`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Studio\_Award` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Studio\_Award` (

`idStudio\_Award` INT NOT NULL,

`Award\_id` INT NOT NULL,

`Studio\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

`is\_winner` BIT NULL,

PRIMARY KEY (`idStudio\_Award`),

CONSTRAINT `fk\_Studio\_Award\_Award1`

FOREIGN KEY (`Award\_id`)

REFERENCES `film\_service\_db`.`Award` (`idAward`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Studio\_Award\_Studio1`

FOREIGN KEY (`Studio\_id`)

REFERENCES `film\_service\_db`.`Studio` (`idStudio`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Studio\_Award\_Award1\_idx` ON `film\_service\_db`.`Studio\_Award` (`Award\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Studio\_Award\_Studio1\_idx` ON `film\_service\_db`.`Studio\_Award` (`Studio\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Person\_Award`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Person\_Award` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Person\_Award` (

`idPerson\_Award` INT NOT NULL,

`Person\_id` INT NOT NULL,

`Award\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

`is\_winner` BIT NULL,

PRIMARY KEY (`idPerson\_Award`),

CONSTRAINT `fk\_Person\_Award\_Person1`

FOREIGN KEY (`Person\_id`)

REFERENCES `film\_service\_db`.`Person` (`idPerson`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Person\_Award\_Award1`

FOREIGN KEY (`Award\_id`)

REFERENCES `film\_service\_db`.`Award` (`idAward`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Person\_Award\_Person1\_idx` ON `film\_service\_db`.`Person\_Award` (`Person\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Person\_Award\_Award1\_idx` ON `film\_service\_db`.`Person\_Award` (`Award\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Festival`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Festival` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Festival` (

`idFestival` INT NOT NULL,

`name` VARCHAR(255) NOT NULL,

`date` DATE NULL,

`description` VARCHAR(255) NULL,

`location` VARCHAR(255) NULL,

PRIMARY KEY (`idFestival`))

ENGINE = InnoDB;

CREATE INDEX `find\_by\_name` ON `film\_service\_db`.`Festival` (`name` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Festival\_Award`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Festival\_Award` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Festival\_Award` (

`idFestival\_Award` INT NOT NULL,

`Festival\_id` INT NOT NULL,

`Award\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idFestival\_Award`),

CONSTRAINT `fk\_Festival\_Award\_Festival1`

FOREIGN KEY (`Festival\_id`)

REFERENCES `film\_service\_db`.`Festival` (`idFestival`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Festival\_Award\_Award1`

FOREIGN KEY (`Award\_id`)

REFERENCES `film\_service\_db`.`Award` (`idAward`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Festival\_Award\_Festival1\_idx` ON `film\_service\_db`.`Festival\_Award` (`Festival\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Festival\_Award\_Award1\_idx` ON `film\_service\_db`.`Festival\_Award` (`Award\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Article`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Article` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Article` (

`idArticle` INT NOT NULL,

`description` VARCHAR(255) NULL,

`title` VARCHAR(255) NOT NULL,

`text\_location` VARCHAR(255) NOT NULL,

`date` DATE NULL,

PRIMARY KEY (`idArticle`))

ENGINE = InnoDB;

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

-- Table `film\_service\_db`.`Film\_in\_Article`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_in\_Article` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_in\_Article` (

`Article\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Article\_id`, `Film\_id`),

CONSTRAINT `fk\_Film\_in\_Article\_Article1`

FOREIGN KEY (`Article\_id`)

REFERENCES `film\_service\_db`.`Article` (`idArticle`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_in\_Article\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_in\_Article\_Film1\_idx` ON `film\_service\_db`.`Film\_in\_Article` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Person\_in\_Article`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Person\_in\_Article` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Person\_in\_Article` (

`Article\_id` INT NOT NULL,

`Person\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Article\_id`, `Person\_id`),

CONSTRAINT `fk\_Person\_in\_Article\_Article1`

FOREIGN KEY (`Article\_id`)

REFERENCES `film\_service\_db`.`Article` (`idArticle`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Person\_in\_Article\_Person1`

FOREIGN KEY (`Person\_id`)

REFERENCES `film\_service\_db`.`Person` (`idPerson`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Person\_in\_Article\_Person1\_idx` ON `film\_service\_db`.`Person\_in\_Article` (`Person\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Studio\_in\_Article`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Studio\_in\_Article` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Studio\_in\_Article` (

`Article\_id` INT NOT NULL,

`Studio\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Article\_id`, `Studio\_id`),

CONSTRAINT `fk\_Studio\_in\_Article\_Article1`

FOREIGN KEY (`Article\_id`)

REFERENCES `film\_service\_db`.`Article` (`idArticle`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Studio\_in\_Article\_Studio1`

FOREIGN KEY (`Studio\_id`)

REFERENCES `film\_service\_db`.`Studio` (`idStudio`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Studio\_in\_Article\_Studio1\_idx` ON `film\_service\_db`.`Studio\_in\_Article` (`Studio\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Cinema`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Cinema` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Cinema` (

`idCinema` INT NOT NULL,

`name` VARCHAR(255) NOT NULL,

`address` VARCHAR(255) NOT NULL,

`phone\_number` VARCHAR(255) NULL,

`description` VARCHAR(255) NULL,

`website` VARCHAR(255) NULL,

PRIMARY KEY (`idCinema`))

ENGINE = InnoDB;

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

-- Table `film\_service\_db`.`Film\_Session`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_Session` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_Session` (

`idFilm\_Session` INT NOT NULL,

`Cinema\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`screening\_date` DATETIME NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idFilm\_Session`),

CONSTRAINT `fk\_Film\_Session\_Cinema1`

FOREIGN KEY (`Cinema\_id`)

REFERENCES `film\_service\_db`.`Cinema` (`idCinema`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_Session\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_Session\_Cinema1\_idx` ON `film\_service\_db`.`Film\_Session` (`Cinema\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Film\_Session\_Film1\_idx` ON `film\_service\_db`.`Film\_Session` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Film\_Festival`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_Festival` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_Festival` (

`idFilm\_Festival` INT NOT NULL,

`Festival\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idFilm\_Festival`),

CONSTRAINT `fk\_Film\_Festival\_Festival1`

FOREIGN KEY (`Festival\_id`)

REFERENCES `film\_service\_db`.`Festival` (`idFestival`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_Festival\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_Festival\_Festival1\_idx` ON `film\_service\_db`.`Film\_Festival` (`Festival\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Film\_Festival\_Film1\_idx` ON `film\_service\_db`.`Film\_Festival` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Photo`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Photo` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Photo` (

`idPhoto` INT NOT NULL,

`description` VARCHAR(255) NULL,

`photo\_location` VARCHAR(255) NOT NULL,

`title` VARCHAR(255) NOT NULL,

PRIMARY KEY (`idPhoto`))

ENGINE = InnoDB;

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

-- Table `film\_service\_db`.`Film\_Photo`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_Photo` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_Photo` (

`Photo\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Photo\_id`, `Film\_id`),

CONSTRAINT `fk\_Film\_Photo\_Photo1`

FOREIGN KEY (`Photo\_id`)

REFERENCES `film\_service\_db`.`Photo` (`idPhoto`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_Photo\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_Photo\_Film1\_idx` ON `film\_service\_db`.`Film\_Photo` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Person\_Photo`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Person\_Photo` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Person\_Photo` (

`Photo\_id` INT NOT NULL,

`Person\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Photo\_id`, `Person\_id`),

CONSTRAINT `fk\_Person\_Photo\_Photo1`

FOREIGN KEY (`Photo\_id`)

REFERENCES `film\_service\_db`.`Photo` (`idPhoto`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Person\_Photo\_Person1`

FOREIGN KEY (`Person\_id`)

REFERENCES `film\_service\_db`.`Person` (`idPerson`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Person\_Photo\_Person1\_idx` ON `film\_service\_db`.`Person\_Photo` (`Person\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Studio\_Photo`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Studio\_Photo` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Studio\_Photo` (

`Photo\_id` INT NOT NULL,

`Studio\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Photo\_id`, `Studio\_id`),

CONSTRAINT `fk\_Studio\_Photo\_Photo1`

FOREIGN KEY (`Photo\_id`)

REFERENCES `film\_service\_db`.`Photo` (`idPhoto`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Studio\_Photo\_Studio1`

FOREIGN KEY (`Studio\_id`)

REFERENCES `film\_service\_db`.`Studio` (`idStudio`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Studio\_Photo\_Studio1\_idx` ON `film\_service\_db`.`Studio\_Photo` (`Studio\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Soundtrack`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Soundtrack` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Soundtrack` (

`idSoundtrack` INT NOT NULL,

`Film\_id` INT NOT NULL,

`soundtack\_location` VARCHAR(255) NULL,

`description` VARCHAR(255) NULL,

`time\_in\_film` INT NULL,

PRIMARY KEY (`idSoundtrack`),

CONSTRAINT `fk\_Soundtrack\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Soundtrack\_Film1\_idx` ON `film\_service\_db`.`Soundtrack` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Franchise`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Franchise` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Franchise` (

`idFranchise` INT NOT NULL,

`name` VARCHAR(255) NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idFranchise`))

ENGINE = InnoDB;

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

-- Table `film\_service\_db`.`Film\_in\_Franchise`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_in\_Franchise` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_in\_Franchise` (

`idFilm\_in\_Franchise` INT NOT NULL,

`Franchise\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`number` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idFilm\_in\_Franchise`),

CONSTRAINT `fk\_Film\_in\_Franchise\_Franchise1`

FOREIGN KEY (`Franchise\_id`)

REFERENCES `film\_service\_db`.`Franchise` (`idFranchise`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_in\_Franchise\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_in\_Franchise\_Franchise1\_idx` ON `film\_service\_db`.`Film\_in\_Franchise` (`Franchise\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Film\_in\_Franchise\_Film1\_idx` ON `film\_service\_db`.`Film\_in\_Franchise` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Video`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Video` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Video` (

`idVideo` INT NOT NULL,

`title` VARCHAR(255) NOT NULL,

`description` VARCHAR(255) NULL,

`video\_location` VARCHAR(255) NOT NULL,

PRIMARY KEY (`idVideo`))

ENGINE = InnoDB;

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

-- Table `film\_service\_db`.`Film\_Video`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Film\_Video` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Film\_Video` (

`Video\_id` INT NOT NULL,

`Film\_id` INT NOT NULL,

`name` VARCHAR(255) NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Video\_id`, `Film\_id`),

CONSTRAINT `fk\_Film\_Video\_Video1`

FOREIGN KEY (`Video\_id`)

REFERENCES `film\_service\_db`.`Video` (`idVideo`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Film\_Video\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Film\_Video\_Film1\_idx` ON `film\_service\_db`.`Film\_Video` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Series`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Series` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Series` (

`idSeries` INT NOT NULL,

`Film\_id` INT NOT NULL,

`closing\_date` DATE NULL,

PRIMARY KEY (`idSeries`),

CONSTRAINT `fk\_Series\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Series\_Film1\_idx` ON `film\_service\_db`.`Series` (`Film\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Season`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Season` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Season` (

`idSeason` INT NOT NULL,

`Series\_id` INT NOT NULL,

` name` VARCHAR(255) NULL,

`number` INT NOT NULL,

`start\_date` DATE NULL,

`end\_date` DATE NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idSeason`),

CONSTRAINT `fk\_Season\_Series1`

FOREIGN KEY (`Series\_id`)

REFERENCES `film\_service\_db`.`Series` (`idSeries`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Season\_Series1\_idx` ON `film\_service\_db`.`Season` (`Series\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Episode`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Episode` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Episode` (

`idEpisode` INT NOT NULL,

`Season\_id` INT NOT NULL,

`name` VARCHAR(255) NULL,

`number` INT NOT NULL,

`duration` INT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`idEpisode`),

CONSTRAINT `fk\_Episode\_Season1`

FOREIGN KEY (`Season\_id`)

REFERENCES `film\_service\_db`.`Season` (`idSeason`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Episode\_Season1\_idx` ON `film\_service\_db`.`Episode` (`Season\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Emotion`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Emotion` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Emotion` (

`like` BIT NOT NULL,

PRIMARY KEY (`like`))

ENGINE = InnoDB;

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

-- Table `film\_service\_db`.`User\_Genre`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Genre` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Genre` (

`Genre\_id` INT NOT NULL,

`User\_id` INT NOT NULL,

`emotion` BIT NOT NULL,

PRIMARY KEY (`Genre\_id`, `User\_id`),

CONSTRAINT `fk\_User\_Genre\_Genre1`

FOREIGN KEY (`Genre\_id`)

REFERENCES `film\_service\_db`.`Genre` (`idGenre`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Genre\_User1`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Genre\_Emotion1`

FOREIGN KEY (`emotion`)

REFERENCES `film\_service\_db`.`Emotion` (`like`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_Genre\_User1\_idx` ON `film\_service\_db`.`User\_Genre` (`User\_id` ASC) VISIBLE;

CREATE INDEX `fk\_User\_Genre\_Emotion1\_idx` ON `film\_service\_db`.`User\_Genre` (`emotion` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`User\_Country`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Country` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Country` (

`Country\_id` INT NOT NULL,

`User\_id` INT NOT NULL,

`Emotion\_like` BIT NOT NULL,

PRIMARY KEY (`Country\_id`, `User\_id`),

CONSTRAINT `fk\_User\_Country\_Country1`

FOREIGN KEY (`Country\_id`)

REFERENCES `film\_service\_db`.`Country` (`idCountry`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Country\_User1`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Country\_Emotion1`

FOREIGN KEY (`Emotion\_like`)

REFERENCES `film\_service\_db`.`Emotion` (`like`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_Country\_User1\_idx` ON `film\_service\_db`.`User\_Country` (`User\_id` ASC) VISIBLE;

CREATE INDEX `fk\_User\_Country\_Emotion1\_idx` ON `film\_service\_db`.`User\_Country` (`Emotion\_like` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`User\_Article`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Article` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Article` (

`User\_id` INT NOT NULL,

`Article\_id` INT NOT NULL,

`Emotion\_like` BIT NOT NULL,

PRIMARY KEY (`User\_id`, `Article\_id`),

CONSTRAINT `fk\_User\_Article\_User1`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Article\_Article1`

FOREIGN KEY (`Article\_id`)

REFERENCES `film\_service\_db`.`Article` (`idArticle`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Article\_Emotion1`

FOREIGN KEY (`Emotion\_like`)

REFERENCES `film\_service\_db`.`Emotion` (`like`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_Article\_Article1\_idx` ON `film\_service\_db`.`User\_Article` (`Article\_id` ASC) VISIBLE;

CREATE INDEX `fk\_User\_Article\_Emotion1\_idx` ON `film\_service\_db`.`User\_Article` (`Emotion\_like` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`User\_Film\_Session`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Film\_Session` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Film\_Session` (

`Film\_Session\_id` INT NOT NULL,

`User\_id` INT NOT NULL,

PRIMARY KEY (`Film\_Session\_id`, `User\_id`),

CONSTRAINT `fk\_User\_Film\_Session\_Film\_Session1`

FOREIGN KEY (`Film\_Session\_id`)

REFERENCES `film\_service\_db`.`Film\_Session` (`idFilm\_Session`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Film\_Session\_User1`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_Film\_Session\_User1\_idx` ON `film\_service\_db`.`User\_Film\_Session` (`User\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Person\_on\_Festival`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Person\_on\_Festival` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Person\_on\_Festival` (

`Festival\_id` INT NOT NULL,

`Person\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Festival\_id`, `Person\_id`),

CONSTRAINT `fk\_Person\_Festival\_Festival1`

FOREIGN KEY (`Festival\_id`)

REFERENCES `film\_service\_db`.`Festival` (`idFestival`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Person\_Festival\_Person1`

FOREIGN KEY (`Person\_id`)

REFERENCES `film\_service\_db`.`Person` (`idPerson`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Person\_Festival\_Person1\_idx` ON `film\_service\_db`.`Person\_on\_Festival` (`Person\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Photo\_in\_Article`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Photo\_in\_Article` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Photo\_in\_Article` (

`Article\_id` INT NOT NULL,

`Photo\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Article\_id`, `Photo\_id`),

CONSTRAINT `fk\_Photo\_in\_Article\_Article1`

FOREIGN KEY (`Article\_id`)

REFERENCES `film\_service\_db`.`Article` (`idArticle`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Photo\_in\_Article\_Photo1`

FOREIGN KEY (`Photo\_id`)

REFERENCES `film\_service\_db`.`Photo` (`idPhoto`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Photo\_in\_Article\_Photo1\_idx` ON `film\_service\_db`.`Photo\_in\_Article` (`Photo\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Video\_in\_Article`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Video\_in\_Article` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Video\_in\_Article` (

`Article\_id` INT NOT NULL,

`Video\_id` INT NOT NULL,

`description` VARCHAR(255) NULL,

PRIMARY KEY (`Article\_id`, `Video\_id`),

CONSTRAINT `fk\_Video\_in\_Article\_Article1`

FOREIGN KEY (`Article\_id`)

REFERENCES `film\_service\_db`.`Article` (`idArticle`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Video\_in\_Article\_Video1`

FOREIGN KEY (`Video\_id`)

REFERENCES `film\_service\_db`.`Video` (`idVideo`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Video\_in\_Article\_Video1\_idx` ON `film\_service\_db`.`Video\_in\_Article` (`Video\_id` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`User\_Franchise`

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

DROP TABLE IF EXISTS `film\_service\_db`.`User\_Franchise` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`User\_Franchise` (

`Franchise\_id` INT NOT NULL,

`User\_id` INT NOT NULL,

`Emotion\_like` BIT NOT NULL,

PRIMARY KEY (`Franchise\_id`, `User\_id`),

CONSTRAINT `fk\_User\_Franchise\_Franchise1`

FOREIGN KEY (`Franchise\_id`)

REFERENCES `film\_service\_db`.`Franchise` (`idFranchise`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Franchise\_User1`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_User\_Franchise\_Emotion1`

FOREIGN KEY (`Emotion\_like`)

REFERENCES `film\_service\_db`.`Emotion` (`like`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_User\_Franchise\_Franchise1\_idx` ON `film\_service\_db`.`User\_Franchise` (`Franchise\_id` ASC) VISIBLE;

CREATE INDEX `fk\_User\_Franchise\_User1\_idx` ON `film\_service\_db`.`User\_Franchise` (`User\_id` ASC) VISIBLE;

CREATE INDEX `fk\_User\_Franchise\_Emotion1\_idx` ON `film\_service\_db`.`User\_Franchise` (`Emotion\_like` ASC) VISIBLE;

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

-- Table `film\_service\_db`.`Comment`

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

DROP TABLE IF EXISTS `film\_service\_db`.`Comment` ;

CREATE TABLE IF NOT EXISTS `film\_service\_db`.`Comment` (

`idComment` INT NOT NULL,

`Film\_id` INT NOT NULL,

`User\_id` INT NULL,

`text` VARCHAR(255) NOT NULL,

PRIMARY KEY (`idComment`),

CONSTRAINT `fk\_Comment\_Film1`

FOREIGN KEY (`Film\_id`)

REFERENCES `film\_service\_db`.`Film` (`idFilm`)

ON DELETE NO ACTION

ON UPDATE NO ACTION,

CONSTRAINT `fk\_Comment\_User1`

FOREIGN KEY (`User\_id`)

REFERENCES `film\_service\_db`.`User` (`idUser`)

ON DELETE NO ACTION

ON UPDATE NO ACTION)

ENGINE = InnoDB;

CREATE INDEX `fk\_Comment\_Film1\_idx` ON `film\_service\_db`.`Comment` (`Film\_id` ASC) VISIBLE;

CREATE INDEX `fk\_Comment\_User1\_idx` ON `film\_service\_db`.`Comment` (`User\_id` ASC) VISIBLE;

USE `film\_service\_db`;

DROP function IF EXISTS `film\_service\_db`.`get\_film\_average\_score`;

DELIMITER $$

USE `film\_service\_db`$$

CREATE FUNCTION `get\_film\_average\_score` (film\_id INT)

RETURNS FLOAT

BEGIN

RETURN (SELECT AVG(score) FROM User\_Film

WHERE Film\_id = film\_id);

END$$

DELIMITER ;

USE `film\_service\_db`;

DROP function IF EXISTS `film\_service\_db`.`get\_old\_birth\_date`;

DELIMITER $$

USE `film\_service\_db`$$

CREATE FUNCTION `get\_old\_birth\_date` (person\_id INT)

RETURNS DATE DETERMINISTIC

BEGIN

RETURN (SELECT birth\_date FROM Person

WHERE id = person\_id ORDER BY id DESC LIMIT 1);

END$$

DELIMITER ;

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

-- function get\_old\_death\_date

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

USE `film\_service\_db`;

DROP function IF EXISTS `film\_service\_db`.`get\_old\_death\_date`;

DELIMITER $$

USE `film\_service\_db`$$

CREATE FUNCTION `get\_old\_death\_date` (person\_id INT)

RETURNS DATE DETERMINISTIC

BEGIN

RETURN (SELECT death\_date FROM Person

WHERE id = person\_id ORDER BY id DESC LIMIT 1);

END$$

DELIMITER ;

USE `film\_service\_db`;

DELIMITER $$

USE `film\_service\_db`$$

DROP TRIGGER IF EXISTS `film\_service\_db`.`Person\_BEFORE\_INSERT` $$

USE `film\_service\_db`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `film\_service\_db`.`Person\_BEFORE\_INSERT` BEFORE INSERT ON `Person` FOR EACH ROW

BEGIN

IF NEW.birth\_date > DATE(NOW()) THEN

SET NEW.birth\_date = NULL;

END IF;

END$$

USE `film\_service\_db`$$

DROP TRIGGER IF EXISTS `film\_service\_db`.`Person\_BEFORE\_INSERT\_1` $$

USE `film\_service\_db`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `film\_service\_db`.`Person\_BEFORE\_INSERT\_1` BEFORE INSERT ON `Person` FOR EACH ROW

BEGIN

IF NEW.death\_date > DATE(NOW()) THEN

SET NEW.death\_date = NULL;

END IF;

END$$

USE `film\_service\_db`$$

DROP TRIGGER IF EXISTS `film\_service\_db`.`Person\_BEFORE\_UPDATE` $$

USE `film\_service\_db`$$

CREATE DEFINER = CURRENT\_USER TRIGGER `film\_service\_db`.`Person\_BEFORE\_UPDATE` BEFORE UPDATE ON `Person` FOR EACH ROW

BEGIN

IF NEW.birth\_date > DATE(NOW()) THEN

SET NEW.birth\_date = get\_old\_birth\_date(NEW.idPerson);

END IF;

IF NEW.death\_date > DATE(NOW()) THEN

SET NEW.death\_date = get\_old\_death\_date(NEW.idPerson);

END IF;

END$$

DELIMITER ;

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

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

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