Assignment 3 - Movie Streaming Service Database

Van Provost

A00311818

IOT-1105

Date Created: 3/23/2025, 12:09 PM.
Date Modified: 3/24/2025, 9:39 AM.

1. Database Design

1.1. Tables

For this database, I created 12 tables to create a fully functional and efficient database for a Movie Streaming Service. The tables I created, with keys and datatypes are:

1.1.1. *users*

Name	Data Type	Key	Other Restrictions	Description
user_id	INT	PK	AI, NOT NULL	Primary key for the users table
first_name	VARCHAR(200)		NOT NULL	First name of the user
last_name	VARCHAR(200)		NOT NULL	Last name of the user
email	VARCHAR(255)		NOT NULL, UNIQUE	Email address of the user
phone_number	VARCHAR(35)		NOT NULL	Phone number of the user
user_details_id	INT	FK	NOT NULL	Foreign key pointing to the user_details_id column in the user_details table
subscription_id	INT	FK	NOT NULL	Foreign key pointing to the subscription_id column in the subscriptions table

Design Choices for *users* table

- Kept the amount of columns to a minimum by moving location related information to the *user_details* table.
- Used VARCHAR(255) for *email* to handle all potential email lengths.
- Used VARCHAR(35) for *phone_number* to accommodate international numbers and enforced uniqueness.
- Made the *email* column unique because an email should only be associated with one user, the reason *phone_number* is not unique is to account for multiple users in one household sharing a home phone.

1.1.2. *user_details*

Name	Data Type	Key	Other Restrictions	Description
user_details_id	INT	PK	AI, NOT NULL	Primary key for the user_details table
street_addr	VARCHAR(255)		NOT NULL	Street address of the user
unit_num	VARCHAR(45)			Unit or apartment number of the user
city	VARCHAR(255)		NOT NULL	City that the user resides in
state	VARCHAR(255)		NOT NULL	State/Province that the user resides in
country	CHAR(2)		NOT NULL	Country that the user resides in (ISO 3166-1 alpha-2)
zip_code	VARCHAR(45)			Zip code or postal code of the user

Design choices for the *user_details* table

- Allowed *unit_num* to be nullable since not all users have one.
- Made *zip_code* nullable to account for the fact that some countries do not have any type of postal code, such as Angola.
- Used CHAR(2) for *country* to enforce the ISO 3166-1 alpha-2 standard.

1.1.3. *subscriptions*

Name	Data Type	Key	Other Restrictions	Description
subscription_id	INT	PK	AI, NOT NULL	Primary key for the subscriptions table
plan_type	ENUM('Basic', 'Pro', 'Family', 'Family Pro')		NOT NULL	Subscription plan types
price	DECIMAL(10,2)		NOT NULL	Price of the subscription
terms	ENUM('Monthly', 'Bimonthly', 'Yearly')		NOT NULL	Various payment term types for subscriptions
purchase_date	TIMESTAMP		NOT NULL	The date/time that the

Design choices made for the *subscriptions* table

- Used ENUM for both *plan_type* and *terms* since they will only be using those set values
- Did not add a *payment_due* column since that can be calculated using the *terms* column and the *purchase_date* column

1.1.4. Passwords

Name	Data Type	Key	Other Restrictions	Description
password_id	INT	PK	AI, NOT NULL	Primary key for the passwords table
password_hash	VARCHAR(255)		NOT NULL	The secure hash of the users password
created_at	TIMESTAMP		NOT NULL, DEFAULT=CURRENT_TIMESTAMP	The date the current password was changed
salt	VARCHAR(255)		NOT NULL	The randomized salt added to the users password
user_id	INT	FK	NOT NULL	Foreign key pointing to the user_id column in the users table

Design choices for the *passwords* table

- Set the default value for *created_at* to CURRENT_TIMESTAMP to ensure that if a password is changed or created, the table will reflect the value it was created at.
- Decided to not use a composite key of the *user_id* and *password_hash* for security and performance reasons, so I created the *password_id* column instead.

1.1.5. watch_history

Name	Data Type	Key	Other Restrictions	Description
watch_history_id	INT	PK	AI, NOT NULL	Primary key for the watch_history table
watched_at	TIMESTAMP		NOT NULL	The time the user watched a specific movie
user_id	INT	FK	NOT NULL	Foreign key pointing to the user_id column in the users table
movie_id	INT	FK	NOT NULL	Foreign key pointing to the movie_id column in the movies table

1.1.6. Ratings

Name	Data Type	Key	Other Restrictions	Description
rating_id	INT	PK	AI, NOT NULL	Primary key for the ratings table
rating	ENUM('1', '2', '3', '4', '5')		NOT NULL	5 point rating scale of a specific movie by a specific user
rated_at	TIMESTAMP		NOT NULL	The time a specific user rated a specific movie
review	TEXT			An optional text review that a user can make
user_id	INT	FK	NOT NULL	Foreign key pointing to the user_id column in the users table
movie_id	INT	FK	NOT NULL	Foreign key pointing to the movie_id column in the movies table

Design choices for the *ratings* table

- Used ENUM for the *rating* column since there will only be numbers 1 to 5 for that column
- Allowed *review* to be nullable since most review or rating systems allow a user to simply choose to quickly rate a movie out of 5.

1.1.7. Movies

Name	Data Type	Key	Other Restrictions	Description
movie_id	INT	PK	AI, NOT NULL	Primary key for the movies table
title	VARCHAR(255)		NOT NULL	The title of the movie
release_date	DATE		NOT NULL	The day the movie was officially released
director	VARCHAR(100)		NOT NULL DEFAULT=Unknown	The director of the movie
description	VARCHAR(255)		NOT NULL	The description of the movie
duration	TIME		NOT NULL	The duration of the movie
language	VARCHAR(150)		NOT NULL	The language that the film is originally filmed with
genre_id	INT	FK	NOT NULL	Foreign key pointing to the genre_id column in the genres table

Design choices for the *movies* table
- Used DATE for *release_date* since an exact time data type would be unnecessary
- Used TIME for *duration* since a date data type is unneeded (Ex. TIMESTAMP)
- Made the *director* column NOT NULL, by provided a default value for if the director of the movie is unknown.

1.1.8. Genres

Name	Data Type	Key	Other Restrictions	Description
genre_id	INT	PK	AI, NOT NULL	Primary key for the genres table
genre_name	VARCHAR(100)		NOT NULL, UNIQUE	Unique genre name

Design choices for the *genres* table
- Used UNIQUE for *genre_name* to ensure that genre names are not repeated/duplicated

1.1.9. movie_producers

Name	Data Type	Key	Other Restrictions	Description
movie_producers_id	INT	PK	AI, NOT NULL	Primary key for the movie_producers table
movie_id	INT	FK	NOT NULL	Foreign key pointing to the movie_id column in the movies table
producer_id	INT	FK	NOT NULL	Foreign key pointing to the producer_id column in the producers table

Design choices for the *movie_producers* table
- Created this table due to the fact that most movies have multiple producers, so now it is possible to query for those producers using this table

1.1.10. Producers

Name	Data Type	Key	Other Restrictions	Description
producer_id	INT	PK	AI, NOT NULL	Primary key for the producers table
first_name	VARCHAR(200)		NOT NULL	First name of the producer
last_name	VARCHAR(200)		NOT NULL	Last name of the producer

1.1.11. movie_actors

Name	Data Type	Key	Other Restrictions	Description
movie_actors_id	INT	PK	AI, NOT NULL	Primary key for the movie_actors table
movie_id	INT	FK	NOT NULL	Foreign key pointing to the movie_id column in the movies table
actor_id	INT	FK	NOT NULL	Foreign key pointing to the actor_id column in the actors table

Design choices for the *movie_actors* table
- Created this table with the same design philosophy as the *movie_producers* table, but to find the cast of a movie.

1.1.12. Actors

Name	Data Type	Key	Other Restrictions	Description
actor_id	INT	PK	AI, NOT NULL	Primary key for the actors table
dob	DATE			Date of birth of the actor
first_name	VARCHAR(200)		NOT NULL	First name of the actor
last_name	VARCHAR(200)		NOT NULL	Last name of the actor

Design choices for the *actors table*

- Made *dob* (date of birth) a nullable object since some actors either do not know their exact date of birth or do not officially share that information

1.2. Relationships

1.2.13. Users & User Details

A one-to-one relationship exists between the *users* and *user_details* tables, as each user has a single detailed address record.

1.2.14. Users & Subscriptions

A one-to-one relationship is enforced between the *users* and *subscriptions* tables, ensuring a user has only one active subscription at a time.

1.2.15. Users & Watch History

A one-to-many relationship exists between the *users* table and *watch_history*, as a user can watch multiple movies over time.

1.2.16. Users & Ratings

A one-to-many relationship exists between the *users* and *ratings* tables, allowing users to rate multiple movies. Each rating is stored as a separate record.

1.2.17. Movies & Genres

A one-to-many relationship is maintained between *movies* and *genres*, where each movie belongs to a single genre for simplicity.

1.2.18. Movies & Producers

A many-to-many relationship exists between *movies* and *producers*, implemented using the *movie_producers* table. This allows multiple producers to be associated with a single movie and vice versa.

1.2.19. Movies & Actors

A many-to-many relationship exists between *movies* and *actors*, implemented using the *movie_actors* table. This enables multiple actors to be linked to a single movie and vice versa.

1.3. Normalization

1. **First Normal Form (1NF)**:

- Each column contains atomic values.
- Each row is uniquely identified by a primary key.

2. **Second Normal Form (2NF)**:

- No partial dependencies exist, as all non-key attributes depend on the entire primary key.
- Composite entities like *movie_producers* and *movie_actors* handle many-to-many relationships efficiently.

1. **Third Normal Form (3NF)**:

- Transitive dependencies are removed by separating user address details into user_details.
- Subscription plans and payment terms are stored in a separate table to avoid repetition in the *users* table.

1.4. Assumptions & Special Considerations

- Each user only has one active subscription at a time.
- A movie belongs to just one genre to simplify categorization.
- A user can rate a movie multiple times, but each will be stored under a new record. This allows the database to query for the newest one using *rated_at* if it needed to update the frontend.
- An email can only be used for one subscription.

2. SQL Script

2.5. Create tables

```
-- MySQL Script generated by MySQL Workbench
-- Mon Mar 24 09:10:03 2025
-- Model: New Model    Version: 1.0
-- 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_ZER
O,NO_ENGINE_SUBSTITUTION';

-- -----------------------------------------------------
-- Schema msdb
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `msdb` ;

-- -----------------------------------------------------
-- Schema msdb
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `msdb` DEFAULT CHARACTER SET utf8 ;
USE `msdb` ;

-- -----------------------------------------------------
-- Table `msdb`.`genres`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`genres` ;

CREATE TABLE IF NOT EXISTS `msdb`.`genres` (
  `genre_id` INT NOT NULL AUTO_INCREMENT,
  `genre_name` VARCHAR(100) NOT NULL,
  PRIMARY KEY (`genre_id`),
  UNIQUE INDEX `genre_name_UNIQUE` (`genre_name` ASC) VISIBLE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`movies`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`movies` ;

CREATE TABLE IF NOT EXISTS `msdb`.`movies` (
  `movie_id` INT NOT NULL AUTO_INCREMENT,
  `title` VARCHAR(255) NOT NULL,
  `release_date` DATE NOT NULL,
  `director` VARCHAR(100) NOT NULL,
  `description` VARCHAR(255) NOT NULL,
  `duration` TIMESTAMP NOT NULL,
  `language` VARCHAR(150) NOT NULL,
  `genre_id` INT NOT NULL,
  PRIMARY KEY (`movie_id`),
  INDEX `fk_movies_genres1_idx` (`genre_id` ASC) VISIBLE,
  CONSTRAINT `fk_movies_genres1`
    FOREIGN KEY (`genre_id`)
    REFERENCES `msdb`.`genres` (`genre_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`user_details`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`user_details` ;

CREATE TABLE IF NOT EXISTS `msdb`.`user_details` (
  `user_details_id` INT NOT NULL AUTO_INCREMENT,
```

```sql
  `street_addr` VARCHAR(255) NOT NULL,
  `unit_num` VARCHAR(45) NULL,
  `city` VARCHAR(255) NOT NULL,
  `state` VARCHAR(255) NOT NULL,
  `country` CHAR(2) NOT NULL,
  `zip_code` VARCHAR(45) NULL,
  PRIMARY KEY (`user_details_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`subscriptions`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`subscriptions` ;

CREATE TABLE IF NOT EXISTS `msdb`.`subscriptions` (
  `subscription_id` INT NOT NULL AUTO_INCREMENT,
  `plan_type` ENUM('Basic', 'Pro', 'Family', 'Family Pro') NOT NULL,
  `price` DECIMAL(10,2) NOT NULL,
  `terms` ENUM('Monthly', 'Bimonthly', 'Yearly') NOT NULL,
  `purchase_date` TIMESTAMP NOT NULL,
  PRIMARY KEY (`subscription_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`users`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`users` ;

CREATE TABLE IF NOT EXISTS `msdb`.`users` (
  `user_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(200) NOT NULL,
  `last_name` VARCHAR(200) NOT NULL,
  `email` VARCHAR(255) NOT NULL,
  `phone_number` VARCHAR(35) NOT NULL,
  `user_details_id` INT NOT NULL,
  `subcription_id` INT NOT NULL,
  PRIMARY KEY (`user_id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC) VISIBLE,
  UNIQUE INDEX `phone_number_UNIQUE` (`phone_number` ASC) VISIBLE,
  INDEX `fk_users_user_details1_idx` (`user_details_id` ASC) VISIBLE,
  INDEX `fk_users_subscriptions1_idx` (`subcription_id` ASC) VISIBLE,
  CONSTRAINT `fk_users_user_details1`
    FOREIGN KEY (`user_details_id`)
    REFERENCES `msdb`.`user_details` (`user_details_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_users_subscriptions1`
    FOREIGN KEY (`subcription_id`)
    REFERENCES `msdb`.`subscriptions` (`subscription_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`watch_history`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`watch_history` ;

CREATE TABLE IF NOT EXISTS `msdb`.`watch_history` (
  `watch_history_id` INT NOT NULL AUTO_INCREMENT,
  `watched_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `user_id` INT NOT NULL,
  `movie_id` INT NOT NULL,
  PRIMARY KEY (`watch_history_id`),
  INDEX `fk_watch_history_movies1_idx` (`movie_id` ASC) VISIBLE,
  INDEX `fk_watch_history_users1_idx` (`user_id` ASC) VISIBLE,
  CONSTRAINT `fk_watch_history_movies1`
    FOREIGN KEY (`movie_id`)
    REFERENCES `msdb`.`movies` (`movie_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_watch_history_users1`
    FOREIGN KEY (`user_id`)
    REFERENCES `msdb`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`ratings`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`ratings` ;

CREATE TABLE IF NOT EXISTS `msdb`.`ratings` (
  `rating_id` INT NOT NULL AUTO_INCREMENT,
  `rating` ENUM('1,', '2', '3', '4', '5') NOT NULL,
  `rated_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `review` TEXT NULL,
  `user_id` INT NOT NULL,
  `movie_id` INT NOT NULL,
  PRIMARY KEY (`rating_id`),
  INDEX `fk_ratings_users1_idx` (`user_id` ASC) VISIBLE,
  INDEX `fk_ratings_movies1_idx` (`movie_id` ASC) VISIBLE,
  CONSTRAINT `fk_ratings_users1`
    FOREIGN KEY (`user_id`)
    REFERENCES `msdb`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_ratings_movies1`
    FOREIGN KEY (`movie_id`)
    REFERENCES `msdb`.`movies` (`movie_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`actors`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`actors` ;

CREATE TABLE IF NOT EXISTS `msdb`.`actors` (
  `actor_id` INT NOT NULL AUTO_INCREMENT,
  `dob` DATE NULL,
  `first_name` VARCHAR(255) NOT NULL,
  `last_name` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`actor_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`movie_actors`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`movie_actors` ;

CREATE TABLE IF NOT EXISTS `msdb`.`movie_actors` (
  `movie_actors_id` INT NOT NULL AUTO_INCREMENT,
  `movie_id` INT NOT NULL,
  `actor_id` INT NOT NULL,
  PRIMARY KEY (`movie_actors_id`),
  INDEX `fk_movie_actors_movies_idx` (`movie_id` ASC) VISIBLE,
  INDEX `fk_movie_actors_actors1_idx` (`actor_id` ASC) VISIBLE,
  CONSTRAINT `fk_movie_actors_movies`
    FOREIGN KEY (`movie_id`)
    REFERENCES `msdb`.`movies` (`movie_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_movie_actors_actors1`
```

```
    FOREIGN KEY (`actor_id`)
    REFERENCES `msdb`.`actors` (`actor_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`producers`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`producers` ;

CREATE TABLE IF NOT EXISTS `msdb`.`producers` (
  `producer_id` INT NOT NULL AUTO_INCREMENT,
  `first_name` VARCHAR(255) NOT NULL,
  `last_name` VARCHAR(255) NOT NULL,
  PRIMARY KEY (`producer_id`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`movie_producers`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`movie_producers` ;

CREATE TABLE IF NOT EXISTS `msdb`.`movie_producers` (
  `movie_producer_id` INT NOT NULL,
  `movie_id` INT NOT NULL,
  `producer_id` INT NOT NULL,
  PRIMARY KEY (`movie_producer_id`),
  INDEX `fk_movie_producers_movies1_idx` (`movie_id` ASC) VISIBLE,
  INDEX `fk_movie_producers_producers1_idx` (`producer_id` ASC) VISIBLE,
  CONSTRAINT `fk_movie_producers_movies1`
    FOREIGN KEY (`movie_id`)
    REFERENCES `msdb`.`movies` (`movie_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_movie_producers_producers1`
    FOREIGN KEY (`producer_id`)
    REFERENCES `msdb`.`producers` (`producer_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `msdb`.`passwords`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `msdb`.`passwords` ;

CREATE TABLE IF NOT EXISTS `msdb`.`passwords` (
  `password_id` INT NOT NULL AUTO_INCREMENT,
  `password_hash` VARCHAR(255) NOT NULL,
  `created_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `salt` VARCHAR(255) NOT NULL,
  `user_id` INT NOT NULL,
  PRIMARY KEY (`password_id`),
  INDEX `fk_passwords_users1_idx` (`user_id` ASC) VISIBLE,
  CONSTRAINT `fk_passwords_users1`
    FOREIGN KEY (`user_id`)
    REFERENCES `msdb`.`users` (`user_id`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

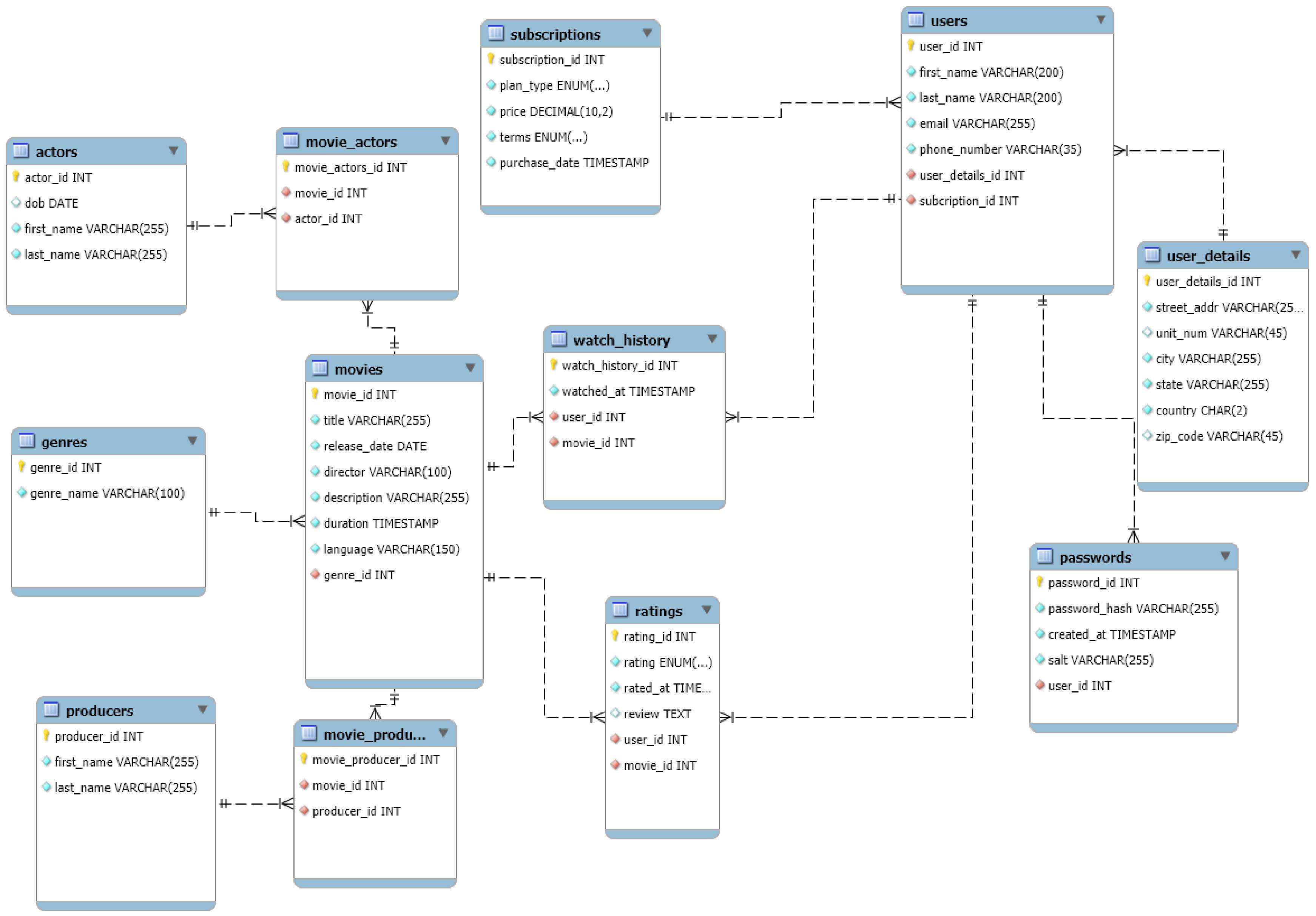
2.6. Insert data (from sample CSV)

This assumes that data for the foreign keys has already been created, this insert doesn't work until data for other tables has also been inserted.

```
USE msdb;

INSERT INTO users (user_id, first_name, last_name, email, phone_number, user_details_id, subscription_id)
VALUES
(1, 'John', 'Sequl', 'jon.sql@outlook.com', '6532220942', 1, 1),
(2, 'Alice', 'Smith', 'alsmith@gmail.com', '5420157924', 2, 2),
(3, 'Van', 'Provost', 'a00311818@mycambrian.ca', '7058892020', 3, 3),
(4, 'Bruce', 'Wayne', 'notbatman123@yahoo.com', '3218750123', 4, 4),
(5, 'Forcht', 'Laest', 'real-person@customdomain.us', '1234567890', 5, 5);
```

3. EER Diagram



4. Sample CSV

```
user_id,first_name,last_name,email,phone_number,user_details_id,_subscription_id
1,John,Sequl,jon.sql@outlook.com,6532220942,1,1
2,Alice,Smith,alsmith@gmail.com,5420157924,2,2
3,Van,Provost,a00311818@mycambrian.ca,7058892020,3,3
4,Bruce,Wayne,notbatman123@yahoo.com,3218750123,4,4
5,Forcht, Laest, real-person@customdomain.us,1234567890,5,5
```