COS221  
Practical 5

Group 9

# Group Members

* Lesedi
* Sechaba
* Tebogo
* Unathi
* Dean

# Task

The project involves collaborating with Hoop, a fictional streaming service, to develop and implement new features for its platform. The goal is to create a user-friendly streaming application that offers a curated collection of movies and TV series worldwide. The application should allow users to easily discover, and filter content based on genres, ratings, release dates, and more. Additionally, users should be able to view detailed information about each title, including cast, crew, plot summaries, and user reviews. Social features such as user profiles, rating, reviewing, and content sharing with friends are also desired.

# Overview

To carry out this task our team will design the required database by first researching the project scenario and creating diagrams to test and review the effectiveness of the database design. We will then create the database and populate it with data from the sources provided.  
The database will hosted on phpMyAdmin using the universities Wheatley server.  
Data will be sourced from the various database’s and Api’s provided and cleaned and augmented as need to fit our own database. We will use python for the data cleaning, augmenting and creation.

# Research

## Technology Trends of the Streaming Industry

## A major trend in the streaming industry is the shift towards microservices architecture, as opposed to monolithic applications. Given that microservices are beyond the scope of this course, we have adhered to a more basic architecture for this practical.

## Most major streaming sites utilize Content Delivery Networks (CDNs). CDNs consist of file servers that store titles as Binary Large Objects (BLOBs). These servers are strategically located worldwide, enabling users to access content from a nearby location, thereby reducing latency. For our practical, we will assume that the file servers are cloud-based applications set up separately from the Hoop web application. Each title can be accessed via a URL called fss\_address, which links to the title stored in the CDN file server relative to the user's region.

## Content Trends of the Streaming Industry

## Streaming platforms such as Apple TV, Netflix, and Hulu invest millions in creating exclusive content. Exclusive content attracts new users since it is not available elsewhere. In recent years, these companies have also acquired the streaming rights to many older, more nostalgic titles to capitalize on the large millennial market. Therefore, it is essential for Hoop to have a good blend of both modern, fresh content and older classics.

## Streaming services must compete with social media platforms for user retention. Encouraging users to share the content they are watching on social media can attract new users to the site, increasing traffic. Social media platforms are also used to market new content and features.

## Movie/TV Show types and genres / Preferences

Some streaming services focus on niche genres, such as BroadwayHD, which specializes in Broadway and musical titles. However, most major platforms aim to offer something for everyone to attract the most users and generate more revenue. Recently, genres like true crime and video game adaptations have increased in popularity, but the top revenue-generating genres remain Adventure, Action, and Comedy. It is important for Hoop to offer a variety of genres.  
Having many genres means having vast amounts of content, which can leave users feeling overwhelmed by the choices. User preferences and recommendations help streaming services present content relevant to users' interests and tastes, mitigating this issue. Netflix uses reinforcement learning algorithms to generate recommendations for users. For Hoop we will simply base recommendations off of watched genres, studio and preferences.

References:  
<https://explodingtopics.com/blog/streaming-trends>

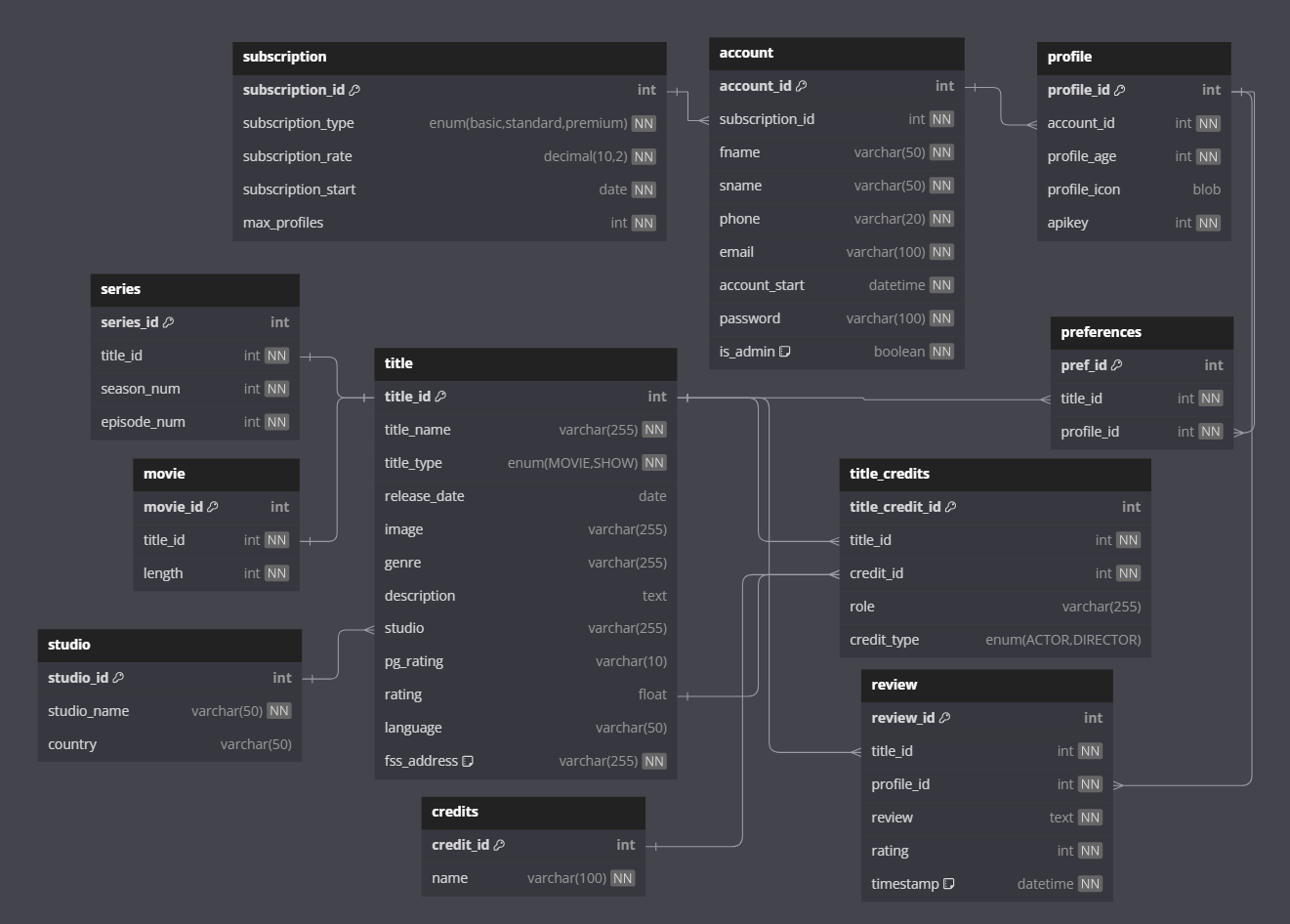
<https://blog.dreamfactory.com/microservices-trends/>

<https://www.makeuseof.com/tag/niche-streaming-services-hate-netflix/>

<https://www.statista.com/statistics/188658/movie-genres-in-north-america-by-box-office-revenue-since-1995/>

<https://research.netflix.com/research-area/recommendations>

# Relational Schema

Diagram:  
  
SQL Code:  
--Creates the schema for the Hoop database

-- Recreate the database if it already exists

DROP DATABASE IF EXISTS COS221\_HOOP;

-- Create Hoop database

CREATE DATABASE COS221\_HOOP;

-- Use Hoop database

USE COS221\_HOOP;

-- Create Tables and Relationships...

-- Create subscription table

CREATE TABLE IF NOT EXISTS subscription (

    subscription\_id INT AUTO\_INCREMENT PRIMARY KEY,

    subscription\_type ENUM('basic', 'standard', 'premium') NOT NULL,

    subscription\_rate DECIMAL(10, 2) NOT NULL,

    subscription\_start DATE NOT NULL,

    max\_profiles INT NOT NULL

);

-- Create account table

CREATE TABLE IF NOT EXISTS account (

    account\_id INT AUTO\_INCREMENT PRIMARY KEY,

    subscription\_id INT NOT NULL,

    fname VARCHAR(50) NOT NULL,

    sname VARCHAR(50) NOT NULL,

    phone VARCHAR(20) NOT NULL,

    email VARCHAR(100) NOT NULL,

    account\_start DATETIME NOT NULL,

    password VARCHAR(100) NOT NULL,

    is\_admin BOOLEAN NOT NULL DEFAULT FALSE,

    FOREIGN KEY (subscription\_id) REFERENCES subscription(subscription\_id)

    ON DELETE CASCADE

);

-- Create profile table

CREATE TABLE IF NOT EXISTS profile (

    profile\_id INT AUTO\_INCREMENT PRIMARY KEY,

    account\_id INT NOT NULL,

    profile\_age INT NOT NULL,

    profile\_icon BLOB,

    apikey INT NOT NULL,

    FOREIGN KEY (account\_id) REFERENCES account(account\_id)

    ON DELETE CASCADE

);

-- Create title table

CREATE TABLE IF NOT EXISTS title (

    title\_id INT AUTO\_INCREMENT PRIMARY KEY,

    title\_name VARCHAR(255) NOT NULL,

    title\_type ENUM('MOVIE', 'SHOW') NOT NULL,

    release\_date DATE,

    image VARCHAR(255),

    genre VARCHAR(255),

    description TEXT,

    studio VARCHAR(255),

    pg\_rating VARCHAR(10),

    rating FLOAT,

    language VARCHAR(50),

    fss\_address VARCHAR(255) NOT NULL DEFAULT "https://www.fss\_address.com",

    FOREIGN KEY (studio) REFERENCES studio(studio\_id)

    ON DELETE CASCADE

);

-- Create series table

CREATE TABLE IF NOT EXISTS series (

    series\_id INT AUTO\_INCREMENT PRIMARY KEY,

    title\_id INT NOT NULL,

    season\_num INT NOT NULL,

    episode\_num INT NOT NULL,

    FOREIGN KEY (title\_id) REFERENCES title(title\_id)

    ON DELETE CASCADE

);

-- Create movie table

CREATE TABLE IF NOT EXISTS movie (

    movie\_id INT AUTO\_INCREMENT PRIMARY KEY,

    title\_id INT NOT NULL,

    length INT NOT NULL,

    FOREIGN KEY (title\_id) REFERENCES title(title\_id)

    ON DELETE CASCADE

);

-- Create credit table

CREATE TABLE IF NOT EXISTS credits (

    credit\_id INT AUTO\_INCREMENT PRIMARY KEY,

    name VARCHAR(100) NOT NULL

);

CREATE TABLE IF NOT EXISTS title\_credits (

    title\_credit\_id INT AUTO\_INCREMENT PRIMARY KEY,

    title\_id INT NOT NULL,

    credit\_id INT NOT NULL,

    role VARCHAR(255),

    credit\_type ENUM('ACTOR', 'DIRECTOR'),

    FOREIGN KEY (title\_id) REFERENCES title(title\_id)

    ON DELETE CASCADE,

    FOREIGN KEY (credit\_id) REFERENCES credits(credit\_id)

    ON DELETE CASCADE

);

-- Create review table

CREATE TABLE IF NOT EXISTS review (

    review\_id INT AUTO\_INCREMENT PRIMARY KEY,

    title\_id INT NOT NULL,

    profile\_id INT NOT NULL,

    review TEXT NOT NULL,

    rating INT NOT NULL,

    timestamp DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

    FOREIGN KEY (title\_id) REFERENCES title(title\_id)

    ON DELETE CASCADE,

    FOREIGN KEY (profile\_id) REFERENCES profile(profile\_id)

    ON DELETE CASCADE

);

-- Create preferances table

CREATE TABLE IF NOT EXISTS preferences (

    pref\_id INT AUTO\_INCREMENT PRIMARY KEY,

    title\_id INT NOT NULL,

    profile\_id INT NOT NULL,

    FOREIGN KEY (title\_id) REFERENCES title(title\_id)

    ON DELETE CASCADE,

    FOREIGN KEY (profile\_id) REFERENCES profile(profile\_id)

    ON DELETE CASCADE

);

CREATE TABLE IF NOT EXISTS studio (

    studio\_id INT AUTO\_INCREMENT PRIMARY KEY,

    studio\_name VARCHAR(50) NOT NULL,

    country VARCHAR(50)

);

--Trigger to ensure max\_profiles is not violated

DELIMITER //

CREATE TRIGGER before\_profile\_insert BEFORE INSERT ON profile

FOR EACH ROW

BEGIN

    DECLARE profile\_count INT;

    DECLARE max\_profiles INT;

    SELECT s.max\_profiles

    INTO max\_profiles

    FROM subscription s

    JOIN account a ON s.subscription\_id = a.subscription\_id

    WHERE a.account\_id = NEW.account\_id;

    SELECT COUNT(\*)

    INTO profile\_count

    FROM profile

    WHERE account\_id = NEW.account\_id;

    IF profile\_count >= max\_profiles THEN

        SIGNAL SQLSTATE '45000'

        SET MESSAGE\_TEXT = 'This account already has the maximum number of profiles.';

    END IF;

END//

DELIMITER ;

# Data Population

To populate our titles and credits data we used the credits.csv and titles.csv files from the provided source: [Netflix Movies and TV Shows (kaggle.com)](https://www.kaggle.com/datasets/dgoenrique/netflix-movies-and-tv-shows)  
This data had to be cleaned and manipulated using python so that it fit the schema of our database.  
The one attribute the title dataset lacked was a poster for each title. This data was populated from the OMDB API: <https://www.omdbapi.com/>

The imdb\_id attribute from titles.csv was used in a get request along with an api key to fetch the poster for each title.  
A python script was also used to assign studios to each title.

Small tables such as the subscription table were manually populated due to the small number of records.

The python scripts used are included in the archive.

# Optimization

A trigger is used to enforce the max\_profiles constraint. This constraint ensures that an account does not create more profiles than their subscription type allows for. The trigger is fired when a user attempts to add a new profile to their account. The function executes a statement to count all records in the profiles table where the account\_id matches that of the user attempting to add a user. If the returned value is less than the max\_profiles of the accounts subscription type, the profile is add. If not, a message is show to the user : 'This account already has the maximum number of profiles.'

This process could be optimized by keeping track of the current number of profiles in an attribute num\_profiles in the account table. This would enable the function to simply read a single value and compare to max\_profiles instead of performing a COUNT() on the entire profile table.