Problem Statement – **nothing changed**

The idea behind my project is to have a command-line program that a user can search for a movie or a tv show directly or through a list corresponding to a steaming site that is currently streaming it. These movies or shows will have general descriptors, including but limited to the people that worked on it (actors, directors, producers, etc.) , the release date, and the genre. Also, each movie and show can be selected to show which streaming sites are currently showing it along with the plan that allows someone to watch and the price of the plan. A database system would be essential because a goal of this project is to have all movies and shows on major streaming sites included in a database or at least have the capability to do so, and in order to achieve this with a smooth and effective end result will require a database system.

Conceptual Database Design – **nothing changed**

Diagram

Description automatically generated

Description:  
Movies and Seasons are streamed on a streaming site through a plan (buy the movie, subscription, etc.).  
Shows have certain descriptors that apply to its seasons as a whole, and a show must consist of seasons and these seasons can be individually streamed on streaming\_sites and have the potential for different people to work on each season.  
People work on movies and seasons and are described by the role they served (actor, director, etc.).

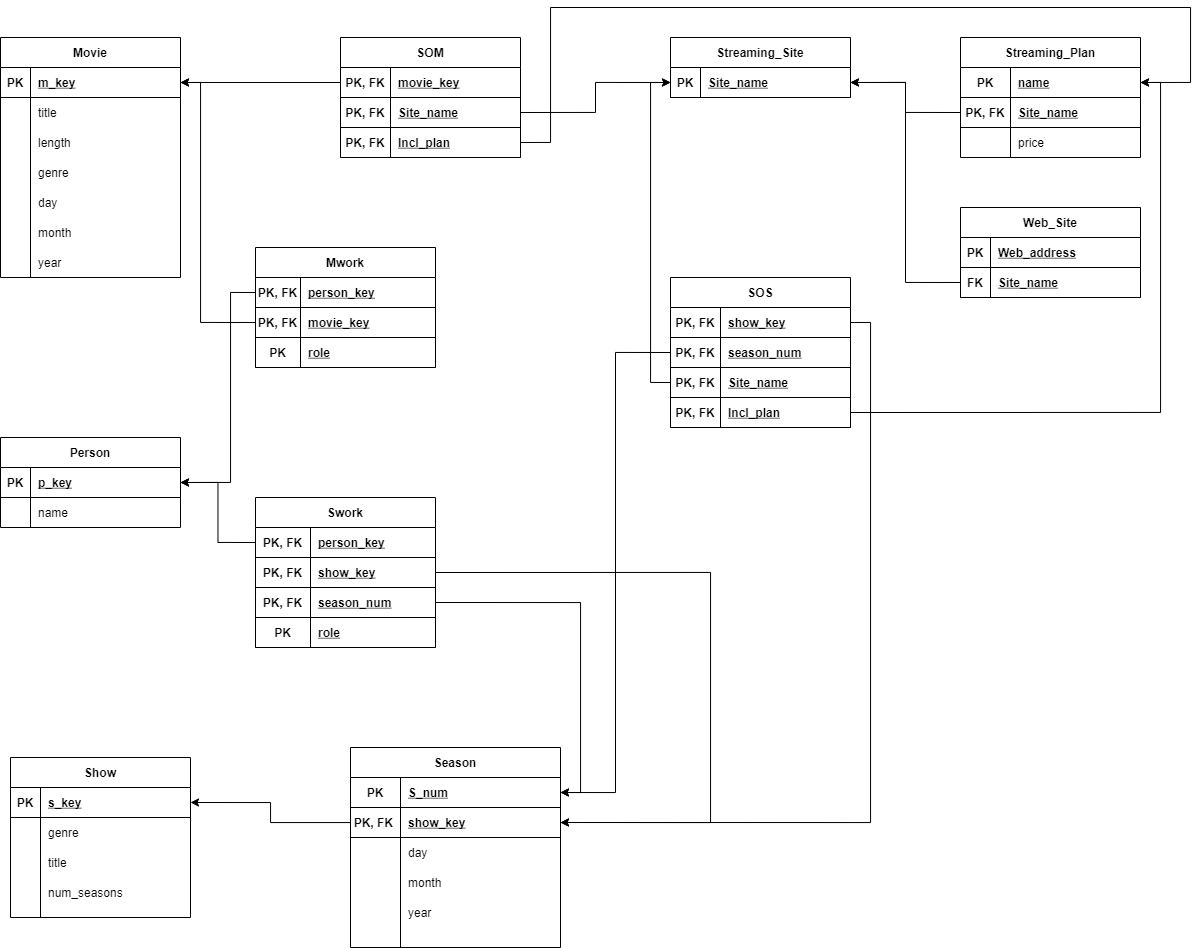
Assumptions:

A streaming\_site must have Seasons of a show and movies

Movies and Shows cannot be differentiated between by title and release\_date

Every streaming\_site has a unique name (This is true in reality)

Logical Database Design: **Not Changed**



Attribute Table: **Changed**

|  |  |  |  |
| --- | --- | --- | --- |
| Relation and attribute name | data type | meaning | integrity constraints |
| Movie : m\_key | int | uniquely identifies a movie | Key, Entity |
| Movie : title | string | title of the movie | None |
| Movie : length | int | the length of the movie in minutes | None |
| Movie : genre | string | the genre of the movie | None |
| Movie : day | int | the day of the month the movie was released on | None |
| Movie : month | int | the month the movie was released on | None |
| Movie : year | int | the year the movie was released on | None |
| SOM : movie\_key | int | identifies the movie in the relationship | Referential, Key, Entity |
| SOM : Site\_name | string | identifies the site in the relationship | Referential, key, entity |
| SOM : Incl\_plan | string | identifies the plan the users can watch the referenced movie on the referenced site on | Referential, key, entity |
| Streaming\_Site : Site\_name | string | the actual streaming site name, ex. Netflix, Hulu. | Key, entity |
| Streaming\_Plan : name | string | the name of the plan | Key, entity |
| Streaming\_Plan : Site\_name | string | references the site\_name for which the plan is under | Referential, key, entity |
| Streaming\_Plan : price | float | the price of the plan | None |
| Web\_Site : Web\_address | string | the actual web address for the streaming site, ex. www.Netflix.com | Key, entity |
| Web\_Site : Site\_name | string | references the site\_name for which the web address links to | Referential |
| Show : s\_key | int | uniquely identifies a show | key, entity |
| Show : genre | string | the genre of the show | none |
| Show : title | string | the title of the show | none |
| Show : num\_seasons | int | the number of seasons the show currently has released | none |
| Season : S\_num | int | the number of the season for the show | Key, entity |
| Season : Show\_key | int | references the show for which the season belongs to | Referential, key, entity |
| Season : day | int | the day the season was released on | none |
| Season : month | int | the month the season was released on | none |
| Season : year | int | the year the season was released on | none |
| Person : p\_key | int | uniquely identifies the person | Key, entity |
| Person : name | string | the name of the person | none |
| Mwork : person\_key | int | identifies the person in the relationship | Referential, key, entity |
| Mwork : movie\_key | int | identifies the movie in the relationship | Referential, key, entity |
| Mwork : role | string | the role the person contributes to movie | Key, entity |
| Swork : person\_key | int | identifies the person in the relationship | Referential, key, entity |
| Swork : show\_key | int | identifies the show in the relationship | Referential, key, entity |
| Swork : season\_num | int | identifies the season of the show that the person worked on | Referential, key, entity |
| Swork : role | string | the role the person contributes to the season | Key, entity |
| SOS : show\_key | int | identifies the show in the relationship | Referential, key, entity |
| SOS : season\_num | int | identifies the season of the show that is being streamed | Referential, key, entity |
| SOS : Site\_name | string | identifies the site that the season is being streamed on | Referential, key, entity |
| SOS : Incl\_plan | string | identifies the plan the users can watch the referenced season on the site being referenced | Referential, key, entity |

Application program design: **Changed**

Function 1. Search for a movie or show  
// accesses “Movie”, and “Show” tables  
Input: movie or show, movie/ show name  
Steps:  
(1) Query the “title” attribute in the respective “movie” or “show” table for matches to the user input  
(2) Display a list of “title” and release dates of close or identical matches for movies, and shows only display the “title”

Function 2. Search through a streaming site  
// accesses “SOS”, “SOM”, “Movie”, “Show” tables  
Input: site name  
Steps:  
(1) Display a list of “titles” and release dates of movies that have its “movie\_key” in a SOM table tuple with “Site\_name” matching the user input, also display a portion of the list as “titles” of shows that have its “show\_key” in a SOS table tuple with “Site\_name” matching the user input

Function 3. Select a show  
// accesses “SOS”, “Season”, “Show”, “Streaming\_Plan”, and “Swork” tables  
Input: selected show of either function 1 or 2  
Steps:  
(1) Display “genre”, “title”, and “num\_seasons” from the selected Show’s table with the matching “s\_key”  
(2) Display each season that has a matching “show\_key” to the selected show’s “s\_key”, along with its release date, “s\_num”  
(3) Display for each season all “site\_name” in an “SOS” tuple with matching “season\_num” and “show\_key” along with the “Incl\_plan” and the associated “price” from the matching “Streaming\_Plan” tuple

Function 4. Select a movie  
// accesses “SOM”, “Movie”, “Streaming\_Plan”, and “Mwork” tables  
Input: selected show of either function 1 or 2  
Steps:  
(1) Display “genre”, “title”, “length” and release date from the “Movie” tuple that has the matching “m\_key” from the user’s selection  
(2) Display all “site\_name” from “SOM” tuples that have a matching “movie\_key”, along with with the “Incl\_plan” and the associated “price” from the matching “Streaming\_Plan” tuple

Function 5. Update  
Steps:  
(1) get input for attributes being requested to change in the tuple  
(2) check if any attributes with the key constraints are changed, and if any are changed check the uniqueness of the new key, if the key is not unique abort  
(3) check if any attributes with the Entity integrity constraint are changed, and if any are changed values are Null, if so abort.  
(3) check if any attributes with the referential integrity constraint are changed, and if any are changed check to see if the new value references an existing tuple, if it does not reference an existing tuple abort  
(4) check if any of the attributes that are referenced by other tuples are changed, if so propagate the change to the other tuples  
(5) update tuple in corresponding table

Function 6. Insert  
Steps:  
(1) get input for all attributes in the corresponding tuple //NULL is an option  
(2) check all attributes with the key constraint to make sure no other tuples in the corresponding table have an identical key, if a table already exists with a key abort  
(3) check all foreign key attributes/ attributes with the referential integrity constraint to ensure that the reference an existing tuple, if no tuple exists with the referenced attribute abort  
(4) check all attributes with the entity integrity constraint that their values in the insert are not NULL, if an attribute in the insert request tuple is NULL and marked with EIC abort  
(5) insert tuple into corresponding table

Function 7. Delete  
Steps:  
(1) Check all attributes in the tuple to see if the are referenced by another tuple, if so abort  
(2) delete the tuple from its corresponding table

Function 8. Total movies and shows  
Input: none  
Steps:  
(1) Sum all “s\_key” from the “Show” table  
(2) Sum all “m\_key” from the “Movie” table  
(3) display the results from steps 1 and 2

Function 9. Show people working on a Movie

Input: m\_key

Steps:

(1) Display every “person” “name” “role” “p\_key” along with the “title” of the “movie” that they worked / that corresponds to the inputted m\_key

Function 10. Show people working on a season of a show

Input: s\_key

Steps:

(1) Display every “person” “name” “role” “p\_key” along with the “title” and all “S\_num” of the “season” of the “show” that they worked / that corresponds to the inputted s\_key

Installation Instructions:

Intended Operating System – Windows 10

Installation Steps:

1. Follow San Yeung’s Installation Guide for MySQL till the end

2. Go to the all ready opened MySQL Workbench/ Open MySQL Workbench and Click the home Icon at the top left of the window

3. Click on Local instance MySQL80

4. Right Click on the navigator on left side of the window (under SCHEMAS) and click Create Schema

5. Name the schema whatever you want and then click apply at the bottom

6. Double click the newly created Schema in the navigator pane on the left

7. extract “MovieDatabase Project.SQL” from “CARLSON\_source.zip”

8. back on MySQL Workbench, click File at the top left corner, then click “Open SQL Script”, find and click on “MovieDatabase Project.SQL” then click open at the bottom right

9. In the center pane of MySQL Workbench, click on the tab labeled “MovieDatabase Project” then click the regular lightning bolt under it

10. the database, with its tables, procedures, and some real data have been installed and can now be used in whatever way

Dependencies – None

User manual:

Once the Database is installed the Movie Database can be used through the MySQL Workbench that it was installed using.

First, click File at the top left then click “New Query Tab”, this tab is what is used to search, add, or make changes to the database.

There are two categories of operations that have already been coded in that can be easily used:

1. Searching – Allows easy access to most of the information in the database

2. Inserting, Updating, and Deleting – Allows for changes to the data inside the database

The remainder of the manual will be used to explain how to execute these operations and what they do.

After typing any of the operations below in their entirety select the entirety of the command  and then click the regular lightning bolt symbol  at the top of the center pane in MySQL Workbench

Category 1. the operations in this category will return with 1 or more query results which will appear on your screen like it does in the image below,

Graphical user interface, application, Word

Description automatically generated

to look through all the information received for the operation executed click on the different “Result #” tabs at the bottom

CALL searchShowMovie ( “movie” or “show”, “title of the movie or show”);

ex.  will search for a movie with the title “The Conjuring” and give more information on any movie in the database with a name like “The Conjuring”

CALL searchStreamingSite ( “site name”);

ex.  will search and then give more information on any movie or show in the database that is being streamed on “Netflix”

CALL totalShowsMovies( ); 

this operation will return how many shows and movies that are in the database, and can only be inputted as shown above

CALL selectMovie(movie\_key for the movie); 

this operation returns information about the selected movie, the best way to get the movie\_key for the movie you want information on is by using the searchShowMovie and finding the movie\_key from that

CALL selectShow(show\_key for the show); 

this operation returns information about the selected show, the best way to get the show\_key for the movie you want information on is by using the searchShowMovie and finding the show\_key from that

CALL workedOnMovie(movie\_key for the movie); 

this operation returns information on the people that worked on the selected movie, the best way to get the movie\_key for the movie you want information on is by using the searchShowMovie and finding the movie\_key from that

CALL workedOnShow(show\_key for the show); 

this operation returns information on the people that worked on the selected show, the best way to get the show\_key for the movie you want information on is by using the searchShowMovie and finding the show\_key from that

Category 2. – this section will be lest descriptive then the previous section

The following will be a least of operations that will either insert new data, update existing date, or delete data stored in the database, each operation will be listed in the way that it needs to be typed in order to work and are designed to be used in conjunction with the section 1 operations

INSERT – these operations add new data, ex. the insertMovie operation will add a new movie to the database, and most things inside the ( ) are required in order to work, so it is recommended to fill the ( ) as much as possible

IMPORTANT NOTE: Not all of these will work with inserting or updating other information

EX. insertStreamingPlan requires an already existing StreamingSite so the insertStreamingSite and insertStreamingPlan must be used in conjunction

CALL insertMovie(“movie title”, movie length, “Genre”, day of release, month of release, year of release);

CALL insertShow(“show title”, “Genre”, number of seasons);

CALL insertSeason(season number, “key of show”, day of release, month of release, year of release); requires existing Show

CALL insertPerson(“person’s name”);

CALL insertStreamingSite(“Streaming Site name”);

CALL insertStreamingPlan( “name of plan”, “name of streaming site”, price); requires existing StreamingSite

CALL insertWebSite(“web address of streaming site”, “streaming site name” ); requires existing StreamingSite

CALL insertShowWork(key of person, key of show, season number, “the role the person played”); requires existing Person and Season

CALL insertMovieWork(key of person, key of movie, “the role the person played” ); requires existing Person and Movie

CALL insertStreamingOnMovie( key of the movie, “site the movie is streaming on”, “name of the plan you can watch the movie on”); requires existing Movie and StreamingSite

CALL insertStreamingOnSeason( key of the show, the specific season number, “site the movie is streaming on”, “name of the plan you can watch the movie on”); requires existing Season and StreamingSite

UPDATE – these operatons update already existing data, ex. the updateMovie operation will update the information on an already existing movie in the database, and all things inside the ( ) are required

IMPORTANT NOTE: If you just want to change one the things in ( ) then you must fill all the other entries with preexisting data, essentially you have to update the whole thing with these operations

CALL updateMovie (key of the movie you want to update, “new title”, new length of movie, “new genre”, new release day, new release month, new release year );

CALL updatePerson (key of the person you want to update, “new name” );

CALL updateSeason (number of the season you want to update, key of show, new release day, new release month, new release year);

CALL updateShow( key of the show you want to update, “new title”, “new genre”, new number of seasons);

CALL updateStreamingPlan (“name of the plan you want to change”, “site name for which this plan is on”, new price);

CALL updateWebSite (“web site you want to change”, “new Streaming site”);

DELETE – these operations delete already existing data, ex. the deleteMovie operation will delete an existing movie from the database, and all things inside the ( ) are required

IMPORTANT NOTE: Some things in the database can not be deleted until the things referencing it are deleted example a show and its season, you can not use CALL deleteShow on a show until you have deleted all of that shows seasons

CALL deleteMovie ( key of the movie you want to delete);

CALL deleteShow ( key of the show you want to delete);

CALL deleteSeason (key of the show the season belongs to, number of the season you want to delete);

CALL deletePerson (key of the person you want to delete);

CALL deleteMovieWork (key of the movie, key of the person, “the role of the person on the movie”);

CALL deleteShowWork (key of the show, season number, key of the person, “the role of the person on that season of the show”);

CALL deleteStreamingSite (“name of the Streaming site you want to delete”);

CALL deleteWebSite ( “web address you want to delete”);

CALL deleteStreamingPlan ( “name of the plan you want to delete”, “Streaming site name related to the plan”);

CALL deleteStreamingOnMovie ( key of the movie, “streaming site name”, “name of the plan”);

CALL deleteStreamingOnSeason ( key of the show, season number, “streaming site name”, “name of the plan”);