Dominique Horard

Term Project – iteration 5

Database Implementation

For this class I would like to implement a database that allows one to track the movies/shows they watch between Netflix, Hulu, and Amazon Prime via a chrome extension. While these streaming sites have their own separate watch histories available for their users to view, it is very cumbersome to locate. These services primarily show “Recently Viewed” somewhere on their site but does not delve into specifics as to the exact date a user consumed a particular piece of content. It would be helpful to have a centralized location to view when and what you have watched, and filter based on the month or year. My intention is to incorporate this later into a fully functional web application, creating this dataset will be the foundation for the other pieces I will develop in other classes in this program.

From personal experience, it can be hard to remember what you have and have not seen. Most of these streaming services do not place a tag on the movies/shows that have been previously seen. I have gotten halfway through a movie only to realize I have seen it a few months ago on a passive watch while doing homework. Having this application would make it significantly easier to doublecheck when and on what site I have seen a movie or show.

For the first use case we have the following:

*Account Signup & Installation*

1. User visits WatchedWhat’s site and installs the extension on their browser.
2. They will need to create an account if they do not have one already.
3. User information is entered on a forms page, which then creates the account in the database.

|  |  |  |
| --- | --- | --- |
| Field | Tracks | Why |
| AccountName | Stores the user created name associated with the account | The user will have the option to log in with their account name or email |
| FirstName | Stores the first name of the account owner | The first and last name will be used within the application where a greeting message such as “Welcome John Doe!” will be displayed |
| LastName | Stores the last name of the account owner | The first and last name will be used within the application where a greeting message such as “Welcome John Doe!” will be displayed |
| EmailAddress | Stores the email of the account owner | This will be stored if emails need to be sent to the user through the application |

Based on the 3rd step in this use case, the most obvious entity that needs to be made is an Account entity to store account information. This entity can be broken down further so information is stored separately, but this will be the main supertype for this use case.

The next use case is as follow:

*Movie/Show Tracking*

1. User goes to a streaming site and starts consuming content.
2. WatchedWhat plugin will track the content watched and collect pertinent information on that content.

|  |  |  |
| --- | --- | --- |
| Field | Tracks | Why |
| StreamingService | Which service a particular piece of content was consumed on | Better distinction for the user, and searching for the content is easier when it can be singled to one service |
| Title | Title of the movie or show | Searching a movie by title is the main way a user would find content |
| YearOfRelease | Year the movie was released | Information will be displayed when the user clicks on the content |
| Runtime | Runtime of the movie in minutes | Information will be displayed when the user clicks on the content |
| WasCompleted | Was the movie/episode of a show completed | Users can see whether the show or movie was completed from within the application once they click on a search result. |

With this use case a couple entities can be identified, namely Streaming Service, and Content. Content will most likely be divided into a Movie and Shows entity later for better distinction in the database. WatchedWhat will track the movies watched and link that to the associated entry in the Account table. This needs to happen for the main purpose of the application to be fulfilled. Structural database rules based on the information gathered are as follows:

1. Content can be watched by many accounts; each account can watch one or more content.

I created this rule because content will not be tracked in the database without having first been watched by an account. It is mandatory that content be associated with one account. The same content can be watched by many different accounts, but I do not know of a clean way that the same piece of content can be linked to many accounts without the use of multiple entity instances. So, until that can be ironed out, content in that entity will be entered as different entity instances that are linked to 1 account. It is optional for an account to be associated with any content. This rule considers that an account will not have any content associated with it when it is first created. Even when they do make the account is created, the user might not use the service going forward. Conversely, an account can have a lot of content associated with it.

1. A streaming service has many content; Many content is associated with one or many streaming services.

This rule dictates that a streaming service has a lot of content that is available to watch. Based on real world examples, this is a common occurrence to the point of being mandatory. It is not a good business model for a new service to come out and only have one show or movie to watch. This would then translate to it not being optional for a lot of content to be on a service. Many streaming services offer original, and exclusive, content or content that appears on their service before others. With that we define that a lot of the same content can be on many services, and a lot of content is exclusive to a service. This relation to services is then made optional for content.

Moving onto the third use case, we have the following:

*Content Tracking/Search*:

1. User signs into WatchedWhat
2. User selects what streaming service they want to return a watch history for, and whether the content is a movie or show.
3. WatchedWhat returns watch history based on the service and can also have them search by date or other criteria such as title, director, or genre.
4. The application pulls matches from the search criteria or the service selected.
5. User selects the specific movie/show.
6. WatchedWhat will bring up the information as it pertains to the movie/show.
7. User can close out of the application.

|  |  |  |
| --- | --- | --- |
| Field | Tracks | Why |
| Director:  FirstName,  LastName,  ContentID | Tracks director’s name, as well as the movie they worked on. | Users will be able to search for the movie based on the director. And the director will be linked to an ID for the content. They can track their most watched director content |
| Actor:  FirstName,  LastName,  ContentID | Tracks actor names and the movie they worked on | Users will be able to search for the movie based on the Actors. And the actor will be linked to an ID for the content. They can track their most watched actor content |
| StudioName | Studio that produced the movie | User will be able to search based on the production company and track their most watched content based on studio |
| Language | Language the movie was made in | User will be able to search based on the language of the content |
| GenreType | Genre(s) the movie is categorized as | User will be able to search based on the content genre and can track their most watched genre |

From this use case new entities need to be created in accordance with steps 3. The user will have the ability to search the content watched based on specific search criteria, and to only track important data as it pertains content, we define the following entities: Studio, Actor, Genre, Director, Language. All popular streaming services (Netflix, Hulu, Prime Video) allow for users to find content based on any of the criteria listed prior. For this database we will want to store that information within their respective entities and associate them with entries in Content.

1. Content is produced by one or more studios; a studio may produce one or many contents.

It is not uncommon for a show or movie to be produced by one or more studios but is it mandatory for a show or movie to be produced by at least one studio. It is also a possibility for a studio that is stored in the database to produce one, or many, pieces of content that is stored as well. A user may watch a movie that is produced by A24, and that studio can then be linked to another movie within the database. This is optional however, as there may be a studio that produces only one piece of content that a user consumes.

1. Content has many actors; an actor can work on one or more content.

Movies and shows will always have many actors on their cast. There is no movie that stars a single actor and no supporting cast. An actor has the option to work on one or more shows/movies. In the context of the database, there may be instances where an actor only appears in one show and the user doesn’t watch another piece of content with that actor, or a user watches many movies with the same actor(s).

1. Content has one or more genres; a genre is assigned to one or more content.

Content will always be categorized in some genre; this is how streaming services organize their show/movies for users to search. It is impossible for a show to not be a comedy, or action, or drama, etc. based on the tone of the content. A genre category will always be assigned to one or more entries in the Content entity as they are stored as the user consumes content.

1. Content is directed by one or more directors; a director can direct one or many content.

A movie or show needs a director, there is no content on these platforms that is devoid of at least one director. And in some cases, a movie will have multiple directors, or a show will have different directors from show to show. Within the database, it will be mandatory for an entry in the Director entity to have at least one associated entry in the Content entity. A user can watch a movie directed by one director, and never watch another movie they make, or be a fan of their work and watch multiple films.

1. Content is created in one language; each language is assigned to one or more content.

A movie or show will always have one main language. What that language is may depend on the director, actors, dialogue, or all 3 things. However, there needs to be a singular language that takes precedence for categorizing the movie on the streaming service. And the language the content is created in can be needs to be assigned to at least on entry in the Content entity but can be assigned to more than one as well.

There is also a structural rule that needs to be defined regarding the Content entity.

1. Content will be a movie, or a show, and never both.

This means it is mandatory and singular for content to be organized by one of these subtypes. There will need to be two subtypes, Movie and Show, to the supertype of Content. The ERD will represent this as a total, mandatory, and disjoint because a show cannot be a movie and vis versa.

Diagram

Description automatically generated

Above is the previous iteration ERD for the entities as they have been defined for the use cases for our database. There is one structural rule which is content will be a movie, or a show, and never both. This is represented by the two entities directly under the Content entity, Movie and Show.

The associative relationships from this version of my conceptual ERD have been further clarified. The entities are better linked to reified linked entities where needed, and entities contain the necessary foreign keys based on the relationship to better discern association. The associative relationships in my conceptual ERD are Studio/Content, Actor/Content, Genre/Content, Language/Content, Director/Content, Account/Content, and StreamingService/Content.

The Studio/Content relationship is N:M. Content is produced by one or more studios; a studio may produce one or many contents.

The Actor/Content relationship is N:M. Content has many actors; an actor can work on one or more content.

The Genre/Content relationship is N:M. Content has one or more genres; a genre is assigned to one or more content.

The Language/Content relationship is 1:M. Content is created in one language; each language is assigned to one or more content.

The Director/Content relationship is N:M. Content is directed by one or more directors; a director can direct one or many content.

The Account/Content relationship is 1:M. Content can be watched by many accounts; each account can watch one or more content.

The StreamingService/Content relationship is N:M. A streaming service has many content; Many content is associated with one or many streaming services.

Below is the previous iteration of my DBMS physical ERD created from the previous conceptual ERD for the relationships listed above.

Diagram

Description automatically generated

I made sure to create an easily identifiable synthetic key for each entity as it is best practices. The synthetic, primary key for all entities is a DECIMAL(12) ID. This allows for many records to be stored in my database as users continue watching content.

Since Studio/Content, Actor/Content, Genre/Content, Director/Content, and StreamingService/Content are N:M relationships, bridging entities were created to better maintain the relationship. These entities are titledStudioContentLink, ActorContentLink, StreamingServiceContentLink, GenreContentLink, DirectorContentLink. These bridging entities have a synthetic key, and foreign keys from the entities they bridge. For the relationships that are 1:M I have placed the foreign key for content in their entities.

With only one specialization-generalization I gave the subtypes for Content, Movie and Show, a primary, foreign, key of content\_id from the Content entity.

The attributes for each entity are outlined as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table** | **Attribute** | **Datatype** | **Reasoning** |
| Account | first\_name | Varchar(255) | This is the first name of the account  holder, up to 255 characters of the  name. |
| Account | last\_name | Varchar(255) | This is the last name of the account  holder, up to 255 characters of the  name |
| Account | email | Varchar(255) | This is the email associated with the account, up to 255 characters of the  email address |
| Account | username | Varchar(255) | This is the username of the account, up to 255 characters of the  username |
| Account | encrypted\_password | Varchar(255) | Accounts need a password. Passwords will be encrypted with salt. 255 characters should be sufficient |
| Actor | first\_name | Varchar(255) | This is the first name of the actor, up to 255 characters of the  name. |
| Actor | last\_name | Varchar(255) | This is the last name of the actor, up to 255 characters of the  name. |
| Content | title | Varchar(1024) | All content will have a title, as it appears on a streaming service. 1024 characters should be enough to accommodate long titles. |
| Content | description | Varchar(1024) | All content will have a description, as it appears on a streaming service. 1024 characters should be enough to accommodate descriptions. |
| AccountContentLink | watched\_date | Date | The date a piece of content was consumed will be tracked here. |
| Director | first\_name | Varchar(255) | This is the first name of the director, up to 255 characters of the  name. |
| Director | last\_name | Varchar(255) | This is the last name of the director, up to 255 characters of the  name. |
| Genre | genre\_name | Varchar(50) | This name of a specific genre, up to 50 characters should be enough. |
| Language | language\_name | Varchar(50) | This name of the language the movie was made in, up to 50 characters should be enough. |
| Movie | runtime\_minutes | Decimal(3) | Movies runtimes are typical an hour or more. Lengthier movies lean towards 2.5 hours. Instead of storing “1hr 30min”, we opt for storing a solely numeric value for simplicity, in this case “90”. Movies running over an hour and 40 minutes will need 3 digits to represent this. |
| Movie | year\_of\_release | Decimal(4) | Storing the full year in requires 4 digits. |
| AccountContentLink | was\_completed | Char(1) | This is a simple way to discern if a movie was completed. 1 is yes, 0 is no. |
| Show | Episode\_title | Varchar(1024) | In addition to the main title, many shows have episode titles of various lengths. 1024 characters should be enough to accommodate long titles. |
| Show | runtime\_minutes | Decimal(2) | Show runtimes are typical an 30 minutes. Lengthier shows lean towards an hour+, but do not exceed the threshold that requires 3 digits to store in minutes. Instead of storing “1hr 30min”, we opt for storing a solely numeric value for simplicity, in this case “90”. Shows will need 2 digits to represent this. |
| Show | year\_of\_release | Decimal(4) | Storing the full year in requires 4 digits. |
| StreamingService | Service\_name | Varchar(50) | This name of the service that the content is on, up to 50 characters should be enough. |
| Studio | Studio\_name | Varchar(255) | This name of the studio that made the content, up to 50 characters should be enough. |
| ActorContentLink | Role\_id | Char(1) | 1 for lead, 0 for anything else. This can be used to display the lead actors when a user clicks into a movie they searched for. |

After reviewing the comments on the last iteration, I have updated my conceptual and physical ERD with the appropriate entities, relationships between bridging entities, and an added a history table. There were also attribute additions to two tables. The conceptual and physical ERDs are included below:

Diagram

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

I have revised the AccountContentLink entity in my physical ERD and repurposed the historical table that was linked to the Content entity to be linked to the AccountContentLink table. The AccountContentLink table contains the information that tracks what account has consumed what content. Below is the physical ERD:

Attributes have been moved from the Movie and Show entities to the AccountContentLink bridging entity, and the account\_id attribute has been removed from the Content entity along with the watch\_date attribute. Watched\_date, minutes\_completed, and was\_complete are now all found in the AccountContentLink entity. The attribute middle\_initial was added with a null value and varchar(1) for special scenarios with actors being best known by their first name, middle initial, and last name such as Michael B. Jordan.

The reasoning behind not adding attributes for start/stop time is that the minutes\_completed attribute can be used against the runtime attribute in the Movie/Show entity to calculate if the content was completed. The time at which a user starts and finishes a movie does not translate to the actual amount of the movie they watched. A user can spend 3 hours watching an hour and a half movie because they paused multiple times.

Adding specific attributes for start and end time is unnecessary if it can be done with a single attribute. I also chose not to add attributes for comments and rating as those are not features that I deem as relevant to this application. The application is only to be used as a means of tracking content watch, it is not an IMDB clone that lets users comment and rate.

All primary and foreign keys have been retained in this ERD, and the attributes defined in the table above have been assigned to the appropriate entity.

I do not see any places where there would be data redundancy in my database looking at the ERD. It appears, thus far, that tables are normalized appropriately. If there are any issues when the database is being created, the ERD can be revised accordingly. The structural rules remain the same. I do not believe there is a new conceptual ERD as there was nothing, I found, that needed to be normalized further. I have included a conceptual ERD that includes the bridging entities to cover all bases.

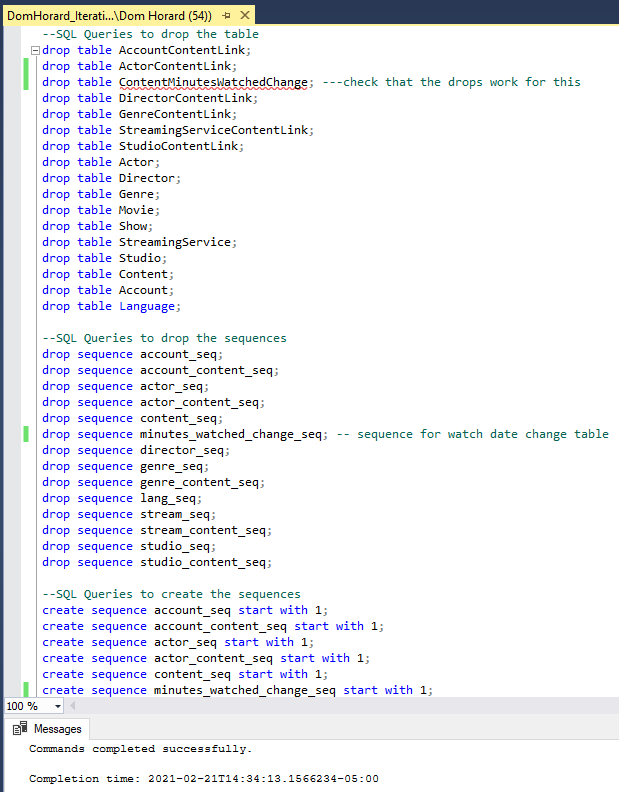
All primary and foreign keys have been retained in this ERD, and the attributes defined in the table above have been assigned to the appropriate entity.

I have also created a history table for the AccountContentLink entity because the most important information to track is when a user has rewatched content and how much of that specific piece of content they watched.

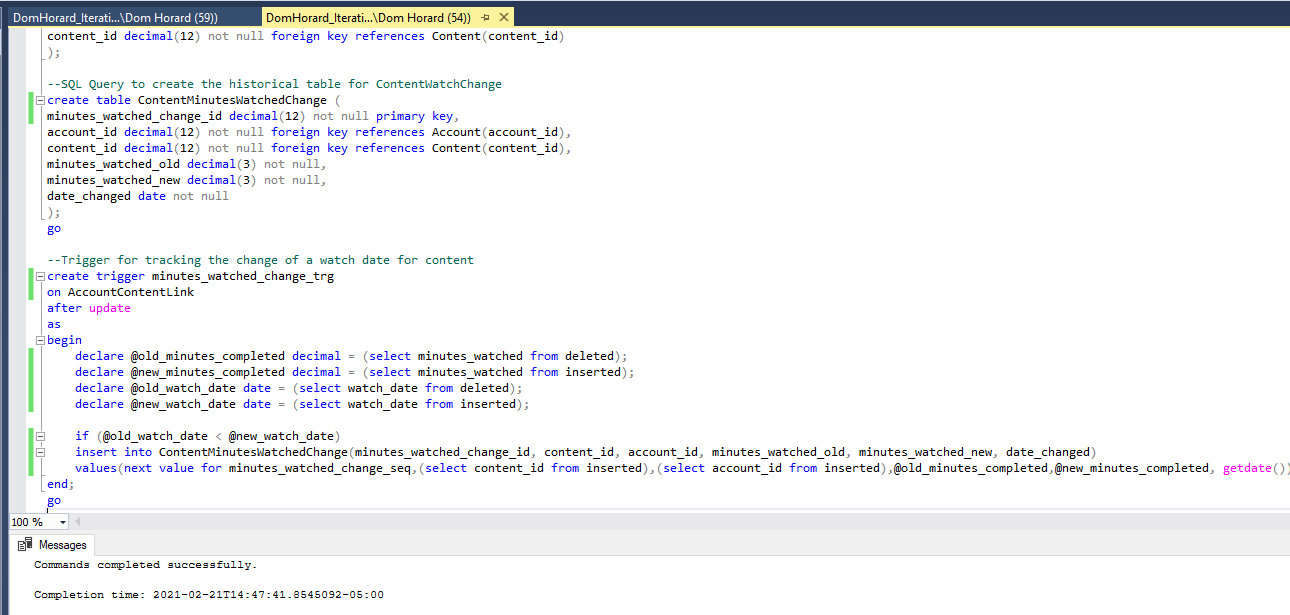
For the historical table, the following attributes can be seen:

|  |  |
| --- | --- |
| Attribute | Description |
| watch\_date\_change\_id | The primary key of the history table. This will be decimal(12) to allow for multiple values |
| minutes\_completed\_old | The old/original minutes completed for a piece of content consumed. Stored as decimal. |
| minutes\_completed\_new | The new minutes completed for a piece of content consumed. Stored as decimal. |
| Watch\_date\_old | The old date that a user watched content |
| Watch\_date\_new | The new date that a user watched content |
| date\_change | The date that this change was made to the database. Datatype of date. |

Below is a screenshot showing the creation and dropping of the tables and sequences for my database:

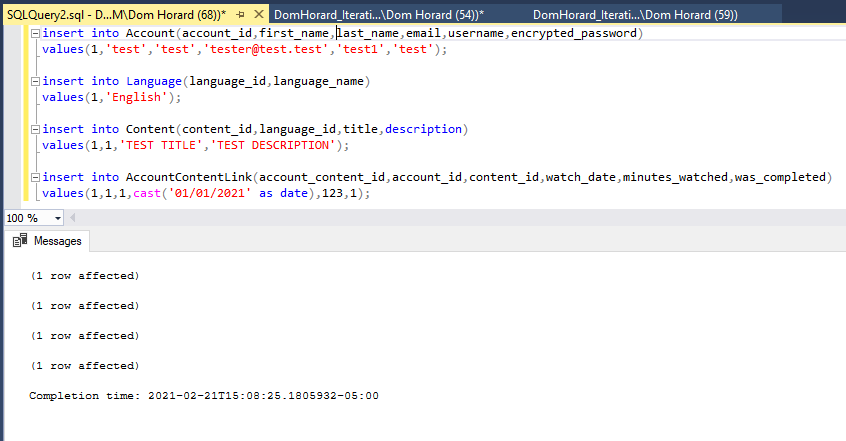


Below is a screenshot of the creation of the history table and trigger:

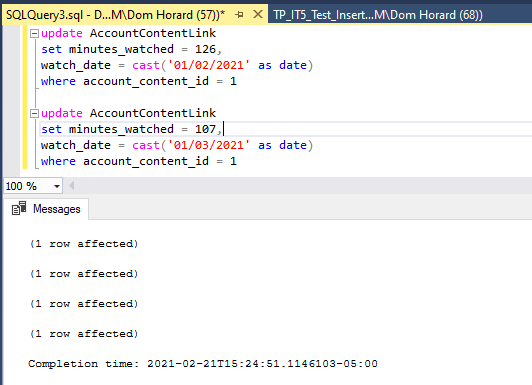


|  |  |
| --- | --- |
| **CODE** | **DESCRIPTION** |
| create trigger minutes\_watched\_change\_trg  on AccountContentLink  after update | Starts the definition for the trigger and names it minutes\_watched\_change\_trg. The trigger is linked to the AccountContentLink table and is executed after any update to that table |
| as  begin | This starts the trigger block |
| declare @old\_minutes\_completed decimal = (select minutes\_watched from deleted);  declare @new\_minutes\_completed decimal = (select minutes\_watched from inserted);  declare @old\_watch\_date date = (select watch\_date from deleted);  declare @new\_watch\_date date = (select watch\_date from inserted); | This saves the old and new minutes completed, as well as the old and new watch date by referencing the deleted and inserted tables in that order |
| if (@old\_watch\_date < @new\_watch\_date) | This checks to make sure that the old date is less than the new date. Because we want to track for the change in watched minutes on different days |
| insert into ContentMinutesWatchedChange(minutes\_watched\_change\_id, content\_id, account\_id, minutes\_watched\_old, minutes\_watched\_new, date\_changed)  values(next value for minutes\_watched\_change\_seq,(select content\_id from inserted),(select account\_id from inserted),@old\_minutes\_completed,@new\_minutes\_completed, getdate()); | This inserts the record into the ContentMinutesWatchedChange table. The primary key for this table is set by using the minutes\_watched\_change\_seq sequence. The old and new minutes completed are taken from the variables, content id and account\_id are taken from the insert statement via a subquery, and the change date is retrieved via getdate function. |
| end; | This ends the trigger definition. |

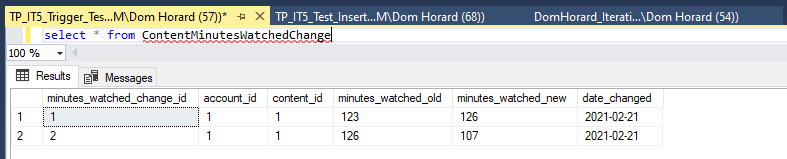
The sequence for this table was created that the top of the query document. Below is the minimum necessary information that needs to be inserted in the table before trying to trigger the trigger:



I updated the AccountContentLink table twice:



Below are the rows that were inserted into the ContentMinutesWatchedChange by the trigger:



Below is a list of the primary keys that have already been indexed:

Content.content\_id

Account.account\_id

Language.language\_id

Genre.genre\_id

Actor.actor\_id

Director.director\_id

Studio.studio\_id

StreamingService.service\_id

ContentWatchChange.watch\_date\_change\_id

AccountContentLink.account\_content\_id

ActorContentLink.actor\_content\_link

DirectorContentLink.director\_content\_id

GenreContentLink.genre\_content\_id

StreamingServiceContentLink.service\_content\_id

StudioContentLink.studio\_content\_id

Below is a table for all the foreign keys that need to be indexed:

|  |  |  |
| --- | --- | --- |
| **Column** | **Unique?** | **Description** |
| Content.account\_id | Not Unique | There can be different content associated for the same account |
| Content.language\_id | Not Unique | There can be different content associated for the same language |
| AccountContentLink.account\_id | Not Unique | The same account can be linked to many content |
| AccountContentLink.content\_id | Not Unique | The same piece of content can be linked to many accounts |
| ActorContentLink.actor\_id | Not Unique | The same actors can be linked to many movies/shows |
| ActorContentLink.content\_id | Not Unique | The same content can be linked to many actors |
| DirectorContentLink.director\_id | Not Unique | The same content can be linked to many content |
| DirectorContentLink.content\_id | Not Unique | The same piece of content can be linked to different directors |
| GenreContentLink.genre\_id | Not Unique | The same genre can be linked to different content |
| GenreContentLink.content\_id | Not Unique | The same content can be linked to different genres |
| StreamingServiceContentLink.service\_id | Not Unique | The same streaming service can be linked to different content |
| StreamingServiceContentLink.content\_id | Not Unique | The same content can be linked to different streaming services |
| StudioContentLink.studio\_id | Not Unique | The same studio can be linked to different content |
| StudioContentLink.content\_id | Not Unique | The same piece of content can be linked to different services |

There are 5 query driven indexes that I have identified based on how the application will be used to retrieve information on a specific movie/show:

Content.title

Actor.first\_name

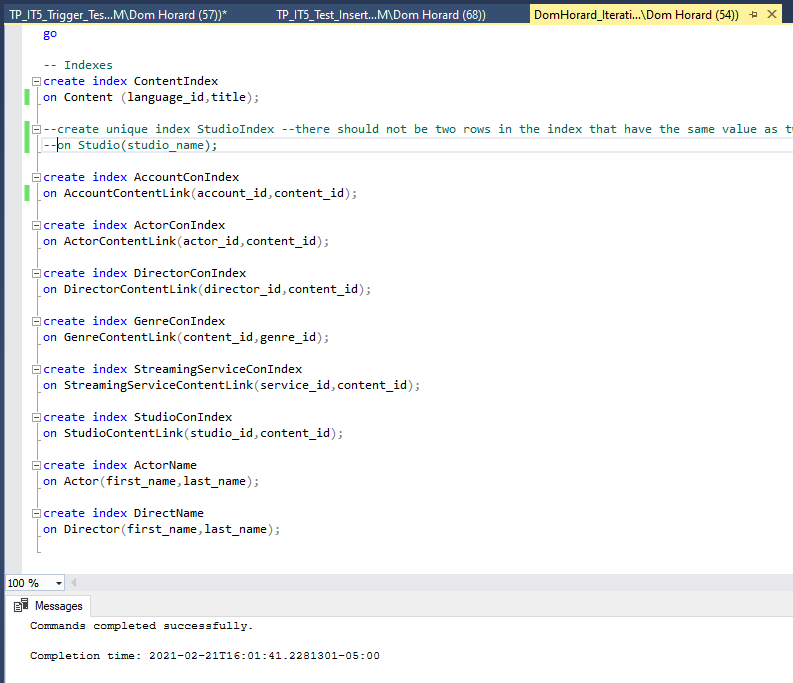
Actor.last\_name

Director.first\_name

Director.last\_name

|  |  |  |
| --- | --- | --- |
| **Column** | **Unique?** | **Description** |
| Content.title | Not Unique | The same content title can be associated with different accounts |
| Actor.first\_name | Not Unique | There can be multiple actors with the same first name |
| Actor.last\_name | Not Unique | There can be multiple actors with the same last name |
| Director.first\_name | Not Unique | There can be multiple directors with the same first name |
| Director.last\_name | Not Unique | There can be multiple directors with the same last name |

These columns are all limiting factors in SQL queries that search what content to display to the user. Content title will be the most common way that a user will search for content. The other columns (actor & director names) are also means by which users will search for their watched content. Below is the execution of the query including the indexes at the bottom of the document.



Below are the stored procedures I created for the following use cases:

*Account Signup & Installation*

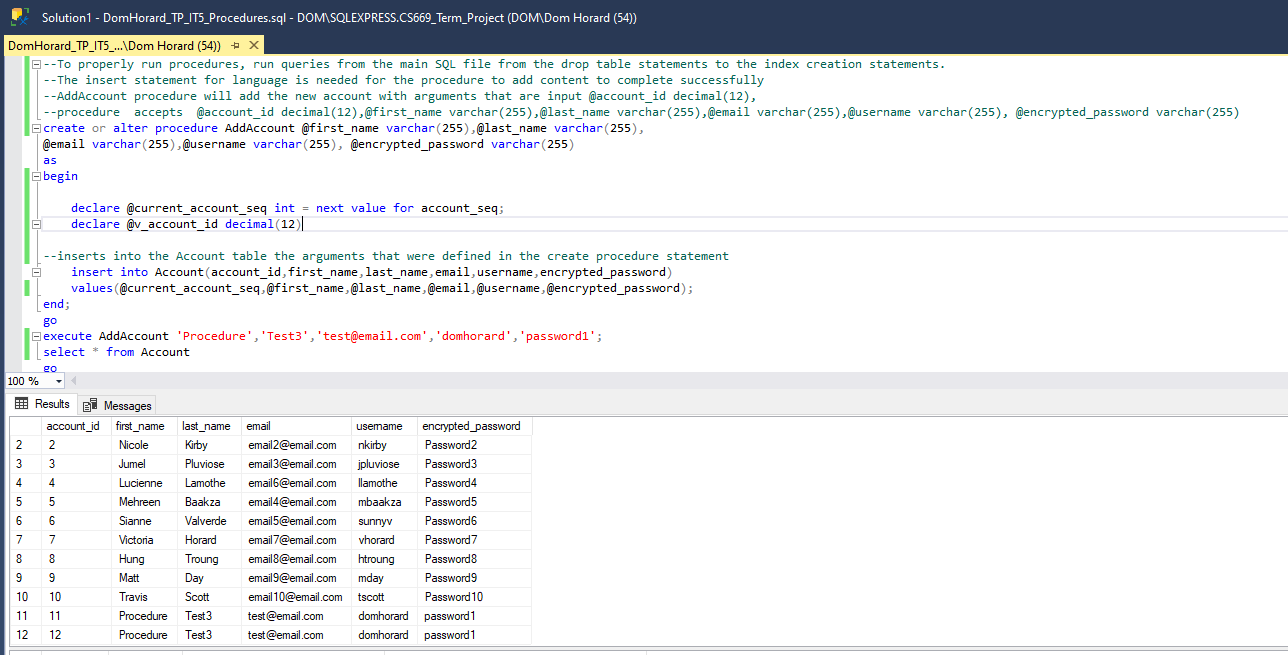
1. User visits WatchedWhat’s site and installs the extension on their browser.
2. They will need to create an account if they do not have one already.
3. User information is entered on a forms page, which then creates the account in the database.

*Movie/Show Tracking*

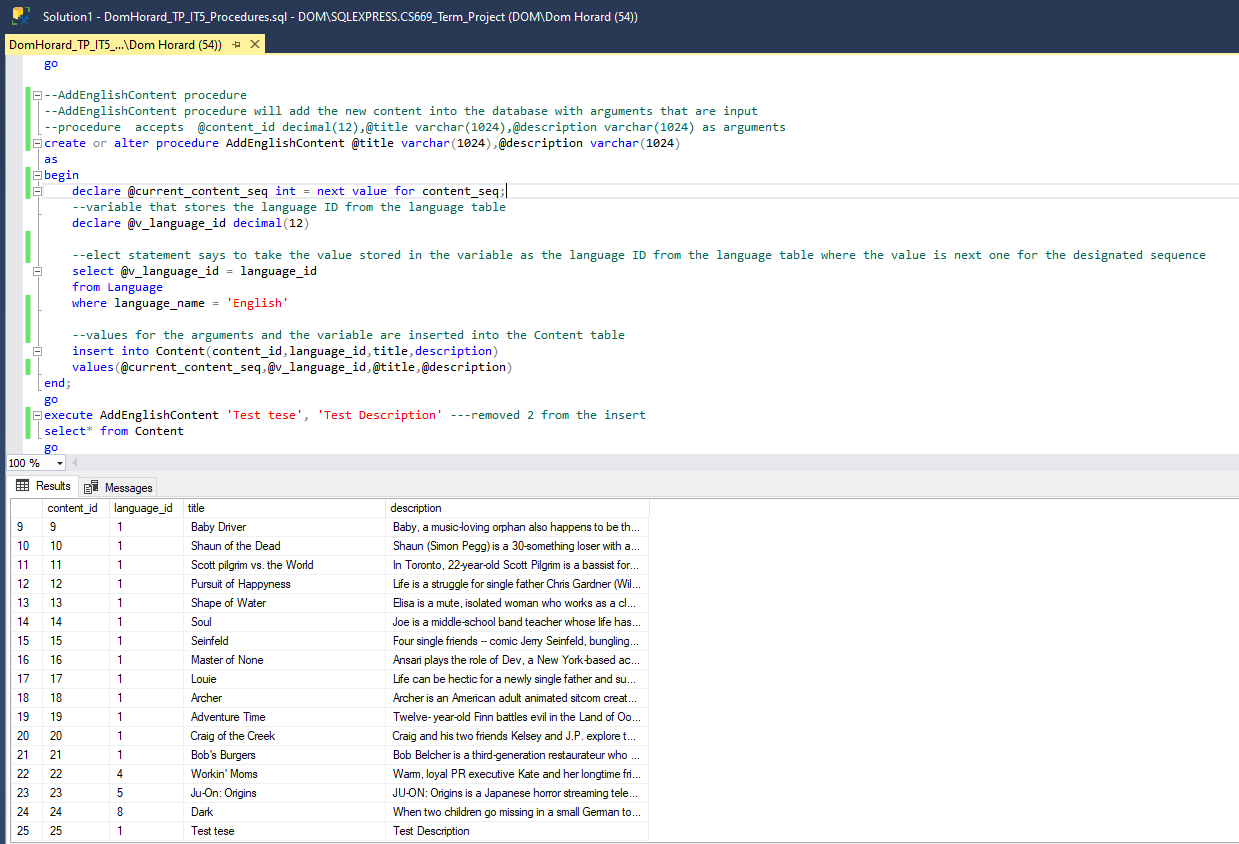
1. User goes to a streaming site and starts consuming content.
2. WatchedWhat plugin will track the content watched and collect pertinent information on that content.

*Content Tracking/Search*:

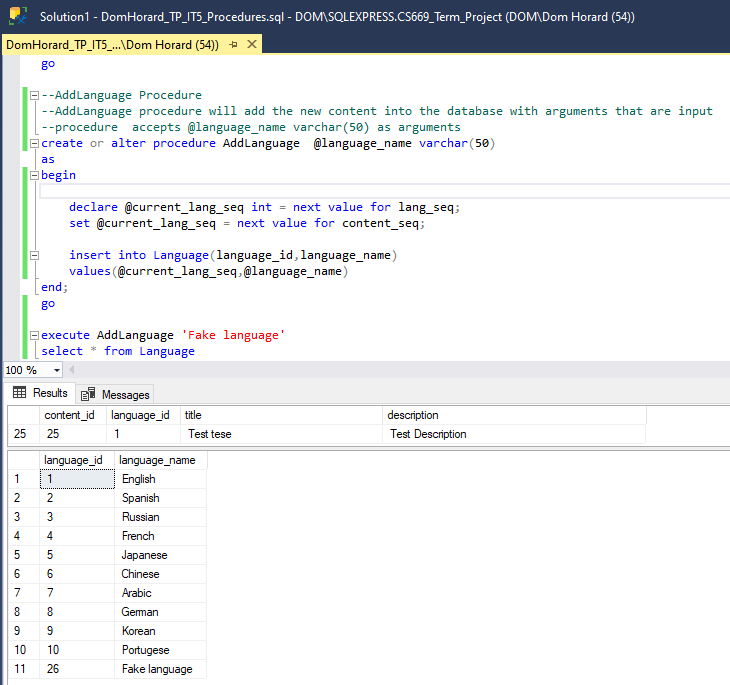
1. User signs into WatchedWhat
2. User selects what streaming service they want to return a watch history for, and whether the content is a movie or show.
3. WatchedWhat returns watch history based on the service and can also have them search by date or other criteria such as title, director, genre, or language.
4. The application pulls matches from the search criteria or the service selected.
5. User selects the specific movie/show.
6. WatchedWhat will bring up the information as it pertains to the movie/show.
7. User can close out of the application.



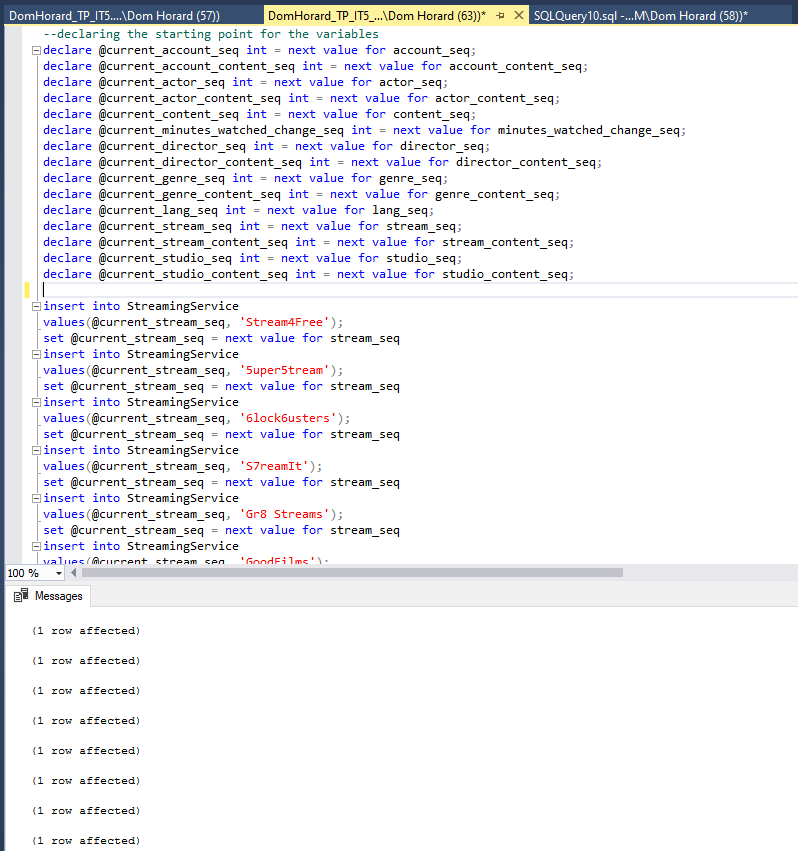
The stored procedure executed above adds a new account to the database accepting arguments for all attributes in that table except the account ID which is created dynamically through the created account\_seq sequence. It then inserts the ID from the sequence and all other arguments into the Account table. The sequence variable needs to be declared within the procedure otherwise there would be no value for the insert statement to insert for that attribute.



The stored procedure executed above adds a new account to the database accepting arguments for all attributes in that table except the content ID which is created dynamically through the created content\_seq sequence. It then inserts the ID from the sequence and all other arguments into the content table. The sequence variable needs to be declared within the procedure otherwise there would be no value for the insert statement to insert for that attribute.



The stored procedure executed above adds a new account to the database accepting arguments for all attributes in that table except the language ID which is created dynamically through the created lang\_seq sequence. It then inserts the ID from the sequence and all other arguments into the language table. The sequence variable needs to be declared within the procedure otherwise there would be no value for the insert statement to insert for that attribute.

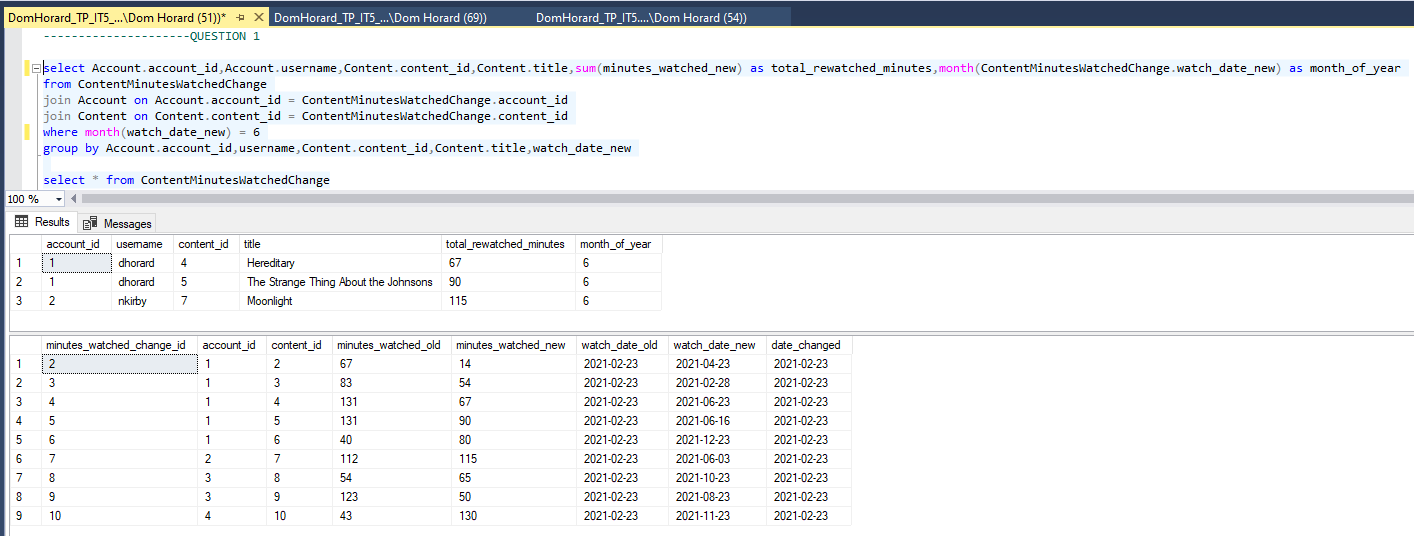


All inserts utilize @current\_sequence\_name to retrieve the primary and foreign keys. All SQL queries for inserting data will be submitted with the document.

I’ve decided to create queries for the following questions:

1. How many minutes of re-watched content was consumed by users in X month? What was the content that was re-watched?

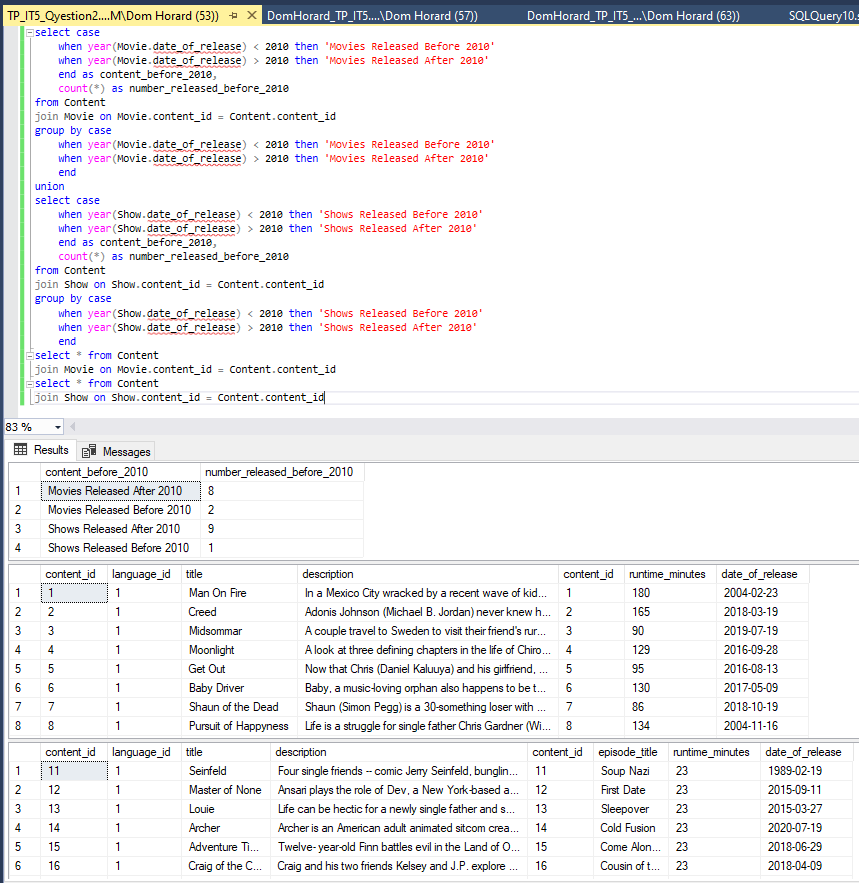
This question is important if I want to analyze how much new content is consumed as opposed to users re-watching the same movie or show again. This can also indicate to the end user what their most watched piece of content is because if the same show/movie is being watched again and again it can be shown to the user what their "favorite content of X month" is.



To make the results easier to read, I joined the Account and Content table to add the username, content, and content\_id columns. I utilized the sum function to add the minutes of re-watched content, and to assign those total minutes to a specific users and the associated content, I grouped by Account.account\_id,username,Content.content\_id,Content.title,watch\_date\_new. I have chosen to return total re-watched minutes for the month of June, included is the history table to validate the variety of data pulled from it.

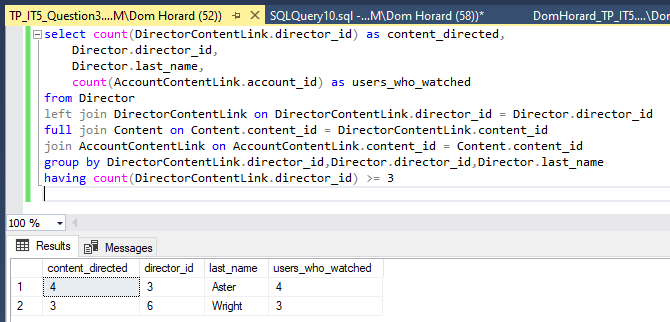
1. How much of the content that was consumed was release before the year 2010, and by who?

This user information could be sold to the streaming services to better help them in purchasing content they deem will do well on their platform. Excluding titles that are classics despite their release year, most users want content that was released relatively recently.



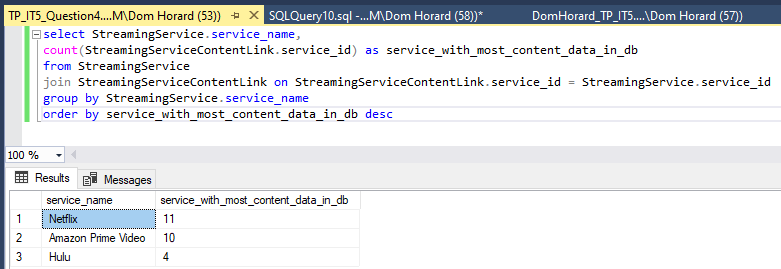
1. What directors have created more than 1 piece of content? And how many people have watched their content?

It is common for services to obtain the rights to stream content from directors that are known in the industry for the quality of there work, and directors that attract the most eyes. For this reason, I created a query that retrieves data that can be sold to streaming services to aid in content buying. They can decide what directors are high value in terms of content purchase. My query counts the directors from the DirectorContentLink entity that appear 3 or more times (This number can be changed in the future if need based on future growth). It presents the director’s ID and last name to identify the director, and how many users the application have watched their content. To retrieve the necessary information to count the users that watched content from that director, I joined the AccountContentLink table through joining the Content table.



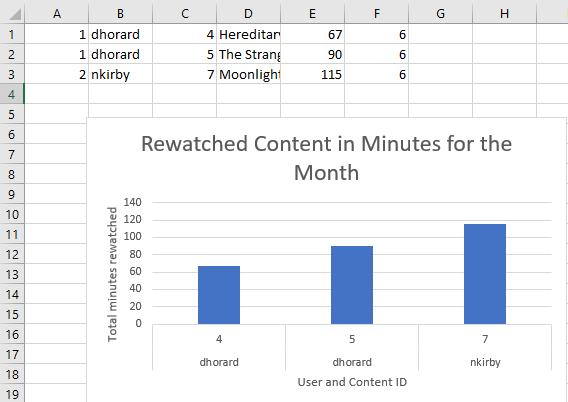
1. Which streaming service does the most content data come from?

The results of this query can be used to better optimize the plugin depending on how big the difference is in terms of content from specific streaming services being stored in the database. If one service is used at a disproportionate amount when compare d to the others, focus could shift to optimizing for that platform first and foremost.



Using the aggregate function count, I was able to get the results for how many times a piece of content with a specific streaming service ID is listed in the bridging entity. I made sure to order by descending order to get the highest value first.

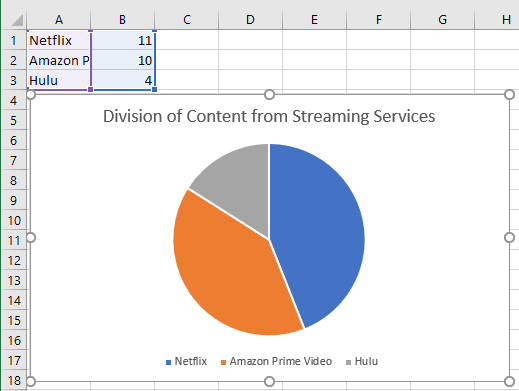
Below is the data visualization for the results of question 1:



The chart shows that there were 2 users in the month of June that re-watched content that was already stored in the database. None of the users had re-watched the same piece of content, so they have different tastes in content with a more thorough review of what content they consumed. This can be verified through a more in-depth investigation of the genre of the content. Pulling in data from the subtype table would also show whether they actually finished the content that was re-watched.

With more information, it could present a more comprehensive view as to what months experience the highest rate of users re-watching content. Inferences could be made as to why users are re-watching content as well. Are these shows/movies they contently revisit despite new content being available on the streaming services? Or is that particularly month known for a lack of new releases? Again, this will go into showing statistics to the user on the frontend of the application for their favorite/most re-watched content. There may also be a benefit to showing this data to a streaming service. If there is a particular piece of content that is constantly being re-watched by multiple users, then they may want to consider renewing the rights to have that content on their platform.

Below is the visualization for question 4:



The chart shows that there is a small percentage of content data residing in the database that comes from Hulu. Amazon Prime and Netflix share a similar split, however. One can infer that Netflix and Amazon Prime are more preferred services than Hulu from this data.

Using this chart, a better sense of which service our users are streaming content from most/least can be understood. Say Hulu remains the lowest streamed from service after a certain period, a decision could be made to either drop support for that service, or shift focus to optimizing for the more popular services that the plugin is used for.

This database is for my WatchedWhat web application, it is going to track the content watched by users via an API used on the chrome browser. A user will watch content on Hulu, Netflix, or Amazon Prime. When a person visits these sites, it is very difficult for them to properly see what content they watched on a particular day. Users may forget if they completed a show/movie, WatchedWhat aims to ameliorate this issue. This database supports user account information, and the necessary information regarding the content being consumed.

The structural database rules, conceptual, and physical ERD for my design outline, as best as I have determined, the proper relationships and entities needed to store this data. I feel confident that I have an adequate design as a jumping off point for this iteration of my database. I can always tweak and expand on these entities if I feel that there are new features and functions that this database can accommodate.

The SQL scripts contain all the table creations that are outlined in the most recent iteration of the physical ERD. Important indexes related to how a user would search, and be presented, their consumed content have been created. Stored procedures have been created and tested for functionality, however, insert statements were the main means by which data was populated into my entities. Useful questions have been identified and implemented to return results that can be used to further the development of the application or turn a profit.

Looking at my database, I did enjoy the process of inputting data. it was the most mechanical action that needed to be performed to realize the usefulness of my project and knowing when/where inserts needed to happen to fulfill the constraints of certain tables was tricky but rewarding once each insert completed without issue. With more time, I would’ve liked to try and make the stored procedures for all of the tables in my database, but trying to figure out the proper way to call a foreign primary key was something I couldn’t manage to accomplish.