Project Phase 2 Final Deliverable

Group S2-13: Meghna Chityala (mchityal) and Om Patel (opatel)

The Database Life-Cycle: Re-engineering Early Stages of YouTube's Database

Introduction

Launched in 2005, YouTube is an online video sharing platform that is headquartered in San Bruno, California. As it is a video-sharing platform, YouTube is dependent on its users, or in other words, its products are its users. Youtube has come far from what it was when it was originally launched; it went from a simple video sharing platform to a platform that now also provides users with subscription plans to tv shows, movie rentals / purchases, sports streaming, and music streaming.

This project aims to re-engineer the early stages of YouTube's Database. The paper first organizes ten developed user stories and a conceptual model using an entity relationship diagram. We then convert the conceptual model to a relational model, determine its functional dependencies, and (possibly) normalize the model. The work included in this paper prepares us for the creation of the physical model where we will populate the database and use SQL queries to interact with the database.

User Stories

ID	Category	As a <role>	I want <goal>	So that <reason>
US1	Complex	As a viewer	I would like to see the most liked content uploads based on each unique genre	so that I can save and use the most popular recipe later for dinner
US2	Complex	As a content creator	I would like to see the most liked content uploads based on each unique genre	so I can base my content uploads on the most recent trends.
US3	Complex	As a viewer	I would like my empty playlists to be deleted	so that I can discard unused playlists and better organize my playlists.
US4	Complex	As a viewer	I would like to remove a cooking video from my personal sports playlist and add it to my personal cooking playlist	because, originally, I accidentally added the video to the wrong playlist; this way I can properly categorize my videos.
US5	Simple	As a content creator	I would like to see my most liked content uploads	so I can create new content that will statistically appease my viewer base.
US6	Analytical	As a business / advertiser	I would like to see the most popular channels sorted by their average views	so I can contact the creators of the channels to advertise my business / product(s) to a huge audience.
US7	Analytical	As a viewer	I would like to see my average Youtube viewing time in minutes	so that I could control how much time I use viewing content on my mobile device.
US8	Something New	As a content creator	I would like to see the top viewed videos uploaded by other content creators	so that I can compete with current trends and see how my competitors are faring.
US9	Simple	As a viewer	I would like to see videos only on cooking Italian cuisine	so I can learn more recipes and view content I am interested in.
US10	Analytical	As a business / advertiser	I would like to view how many people clicked on my companies embedded links	so that I can see how many people engage with the advertisements and my entertainment business

Conceptual Model

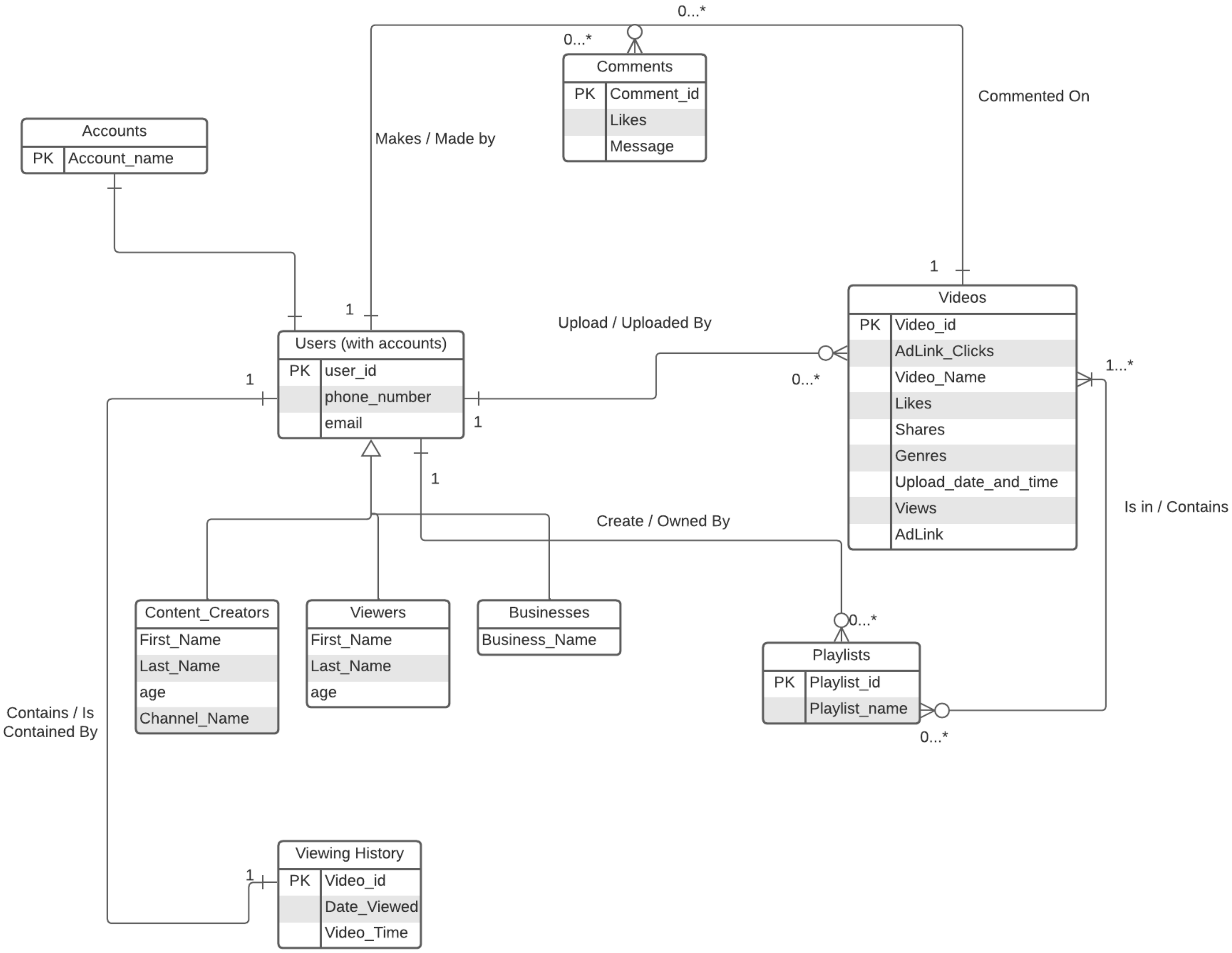

Figure 1: This is a snapshot of the updated conceptual model.

Relational Model

Users (**user_id**, email, phone_number)
Content_Creators (First_Name, Last_Name, age, Channel_Name, **user_id**)
Viewers (First_Name, Last_Name, age, **user_id**)
Businesses (Business_name, **user_id**)
Viewing_History (**Video_id**, Video_time, Date_Viewed, user_id)
Accounts (**Account_name**, **user_id**)
Videos (**Video_id**, Video_Name, Likes, Shares, Genres, Upload_date_and_time, user_id, Views, AdLink, AdLink_Clicks)
Comments (**Comment_id**, Message, Likes, user_id, Video_id)
Playlists (**Playlist_id**, Playlist_name, Video_id)

The many to many relationship between the Playlists table and the Videos table has been accounted for by including the Video_id as a foreign key in the Playlists table (in the relational model above). To account for inheritance, we have included the user_id as a foreign and primary key in the Businesses, Viewers, and Content_Creators tables (as they are various types of users). We also include user_id as a foreign and primary key in the Accounts relation to uniquely pair the account to the user. In the comments relation, we insert the user_id and Video_id as a foreign key to have a specific user who created the comment, and the specific video where the comment is located. The Viewing history also has the associated user_id (foreign key mapping the viewing history to the unique user) and the Videos also have the associated user_id (foreign key) who is labeled as the video creator.

Functional Dependencies

US1:
Video_id → Video_Name, Likes, Shares, Genre, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks
Playlist_id → Playlist_name, Video_id, user_id
User_id → Account_Name, First_Name, Last_Name, age
User_id, Account_Name → email, phone_number

US2:
Video_id → Video_Name, Likes, Shares, Genre, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks
User_id, Account_Name → email, phone_number
User_id → Account_Name, First_Name, Last_Name, age, Channel_Name

US3:
Playlist_id → Playlist_name, Video_id, user_id
User_id → Account_Name, First_Name, Last_Name, age
User_id, Account_Name → email, phone_number

US4:
Playlist_id → Playlist_name, Video_id
User_id → Account_Name, First_Name, Last_Name, age
User_id, Account_Name → email, phone_number
Video_id → Video_Name, Likes, Shares, Genre, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks

US5:
Video_id → User_id
Video_id, User_id → Video_Name, Likes, Shares, Genre, Upload_date_and_time
User_id → Account_Name, First_Name, Last_Name, age, Channel_Name
User_id, Account_Name → email, phone_number

US6:
Video_id → Video_Name, Likes, Shares, Genres, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks
User_id → Account_Name, Business_Name, First_Name, Last_Name, age
User_id, Account_Name → email, phone_number

US7:
Video_id → Video_Name, Likes, Shares, Genres, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks
User_id → Account_Name, First_Name, Last_Name, age
User_id, Account_Name → email, phone_number
Video_id → Video_time, Date_Viewed, user_id

US8:
Video_id → Video_Name, Likes, Shares, Genres, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks
User_id → Account_Name, First_Name, Last_Name, age, Channel_Name
User_id, Account_Name → email, phone_number

US9:
Video_id → Video_Name, Likes, Shares, Genres, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks

US10:
Video_id → Video_Name, Likes, Shares, Genres, Upload_date_and_time, user_id, Views, AdLink, AdLinkClicks
User_id → Account_Name, Business_Name, First_Name, Last_Name, age
User_id, Account_Name → email, phone_number

The following list includes assumptions made about our current model after developing our initial relational model and functional dependencies. We will continue to update the assumptions once we have created our final physical model (post normalization):

1. The email and phone number cannot determine user_id or account_name as one individual can have their email address & phone number attached to multiple accounts
2. In Youtube, account_names are unique (they are distinct)
3. A video can be uploaded by only one user, not by multiple users.
4. Many videos can be contained in many playlists
5. Videos can overlap between playlists; for example, one video can be a part of 2 different playlists
6. Each video can only have 1 genre type
7. There's a possibility that a video does not contain advertisements (ads)
8. If an advertisement is created, it will be on at least one video
9. A playlist does not have to be used

Normalization

The current relational model we have developed above is not completely normalized. We have a many to many relationship between Videos and Playlists entities, non prime attributes determining other attributes, and partial dependencies within a table itself. To convert our relational model to the physical model, we must go through the process of normalization to account for these redundancies. Below outlines our methods for normalizing the above model:

US1:
There is a many to many relationship between Videos and Playlists. We can account for this anomaly by creating an intermediate table called Playlist_Entries, which contains Video_id and Playlist_id combinations to represent which videos are in which playlists. This is a one to many relation from Videos to Playlist_Entries and from Playlist to Playlist_Entries to eliminate the many to many relationship defined before in our conceptual model. Another issue we have encountered with this user story relates to the Genres attribute in the Videos table. Currently, there is a possibility that the same genre can appear more than once in the Videos table, which would introduce the possibility of a transitive dependency. We can resolve this issue by creating a Genres table, containing the Genre_Name and Genre_id (uniquely identifying the genre name) and replacing the Genres attribute in the Videos table. There is now a one to many relation established between Genres and Videos.

US2:
After addressing the two major anomalies discovered when executing the first user story, we realize there are no anomalies occurring in user story 2. This user story is in BCNF because the Videos, Accounts, Users, and Genres tables contain no redundancies and cannot be further decomposed with respect to the functional dependencies. The newly constructed functional dependency for Genres is as follows: Genre_id → Genre_Name and Video_id → Genre_id.

US3:
After addressing the anomaly with Videos and Playlists discovered when executing the first user story, we realize there are no anomalies occurring in user story 3. This user story is in BCNF because the Playlists, Videos, Accounts, and Users tables contain no redundancies and cannot be further decomposed with respect to the functional dependencies.

US4:
After addressing the anomaly with Videos and Playlists discovered when executing the first user story, we realize there are no anomalies occurring in user story 4. This user story is in BCNF because the Playlists, Videos, Accounts, and Users tables contain no redundancies and cannot be further decomposed with respect to the functional dependencies.

US5:
After addressing the two major anomalies when executing the first user story, we realize there are no anomalies occurring in user story 5. This user story is in BCNF because the Videos, Accounts, and Users tables contain no redundancies and cannot be further decomposed with respect to the functional dependencies. However, to account for inheritance, we will create a one to one relationship between Content_Creators and Users in the final physical model (and the same goes for the Businesses and Viewers tables).

US6:
After addressing the two major anomalies when executing the first user story, we realize there are no anomalies occurring in the user story 6. This user story is in BCNF because the Videos, Accounts, and Users tables contain no redundancies and cannot be further decomposed with respect to the functional dependencies.

US7:
Similar to user story 6, after addressing the two major anomalies when executing the first user story, it seems like there are no anomalies occurring in user story 7. However, since each user can watch many videos multiple times, we will make a Viewing_History table to keep track of their viewing history corresponding to each video (based on user_id). The Viewing_History has a Viewing_Timestamp attribute which keeps track of the most recent date and time the user has viewed the video, and the Viewing_Duration sums up the total time the user spends watching that particular video. The Viewing_History table serves as an intermediary between the Videos table and Users table. The newly constructed functional dependency for Viewing_History is as follows: Video_id, User_id → Viewing_Timestamp, Viewing_Duration.

US8:
Now, after addressing three major anomalies earlier, we realize there are no anomalies occurring in the user story 8. This user story is in BCNF because the Videos, Accounts, Views, Content_Creators and Users tables contain no redundancies and cannot be further decomposed with respect to the functional dependencies.

US9:
Similar to user story 8, after addressing three major anomalies earlier, we realize there are no anomalies occurring in the user story 9. This user story is in BCNF because the Videos, Accounts, Views, and Genres tables contain no redundancies and cannot be further decomposed with respect to the functional dependencies.

US10
Similar to user story 9, after addressing the three major anomalies earlier, it seems like there are no anomalies occurring in user story 10. However, one video can contain many ads and one ad can be in many videos. Currently, this exhibits a many to many relationship. We can solve this by using two intermediate tables. By creating an Ad_Clicks table, we can keep track of the number of clicks that the specific advertisement on a unique video receives. Additionally, we can create a second intermediary table called Ads to keep track of the specific Ad_id, Ad_Link, and Ad_Name. To associate the ads with businesses, we connect the Ads table to Users; this way an ad can be associated with an user_id directly. There is now a one to many relationship from Ads to Ad_Clicks, a one to many relationship from Videos to Ad_Clicks, and a one to many relationship from Users to Ads. The newly constructed functional dependency for Ads is as follows: Ad_id → Ad_Link, Ad_Name, User_id and for Ad_Clicks, the newly constructed functional dependency is as follows: Ad_id, Video_id → Num_Clicks.

Our updated relational model is as follows:

Videos (**Video_id**, Video_Name, Likes, Views, Shares, User_id, Genre_id)
Genres (**Genre_id**, Genre_Name)
Playlist_Entries (**Playlist_id**, **Video_id**)
Playlists (**Playlist_id**, Playlist_Name, User_id)
Comments (**Comment_id**, Message, Likes, **User_id**, **Video_id**)
Users (**User_id**, Email, Phone_Number)
Viewers (First_Name, Last_Name, Age, **User_id**)
Content_Creators (First_Name, Last_Name, Age, Channel_Name, **User_id**)
Businesses (Business_Name, **User_id**)
Accounts (**Account_Name**, **User_id**)
Ads (**Ad_id**, Ad_Link, Ad_Name, User_id)
Ad_Clicks (**Ad_id**, **Video_id**, Num_Clicks)
Viewing_History (**Video_id**, **User_id**, Viewing_Timestamp, Viewing_Duration)

Our updated assumptions are as follows (in addition to the ones written earlier):

1. A viewer may or may not have viewed a video
2. Playlist do not determine the genre of the videos in the given playlist and vice versa
3. Comments can be random and do not have to be related to the video the user is commenting on
4. Not all videos have to be in a playlist (a video must not necessarily be in a playlist)
5. Advertisements can only be created by businesses
6. If a user views the same video multiple times, the viewing duration is cumulative, but the timestamp is the most recent timestamp
7. With respect to our user stories, we will disregard video upload timestamp
8. Each video can have zero or more than one advertisement
9. Playlist names do not have to be unique, since the user specifies them
10. The email and phone number cannot determine user_id or account_name as one individual can have their email address & phone number attached to multiple accounts
11. In Youtube, account_names are unique (they are distinct)
12. A video can be uploaded by only one user, not by multiple users.
13. Many videos can be contained in many playlists
14. Videos can overlap between playlists; for example, one video can be a part of 2 different playlists
15. Each video can only have 1 genre type
16. There’s a possibility that a video does not contain advertisements (ads)
17. If an advertisement is created, it will be on at least one video
18. A playlist does not have to be used

Physical Model

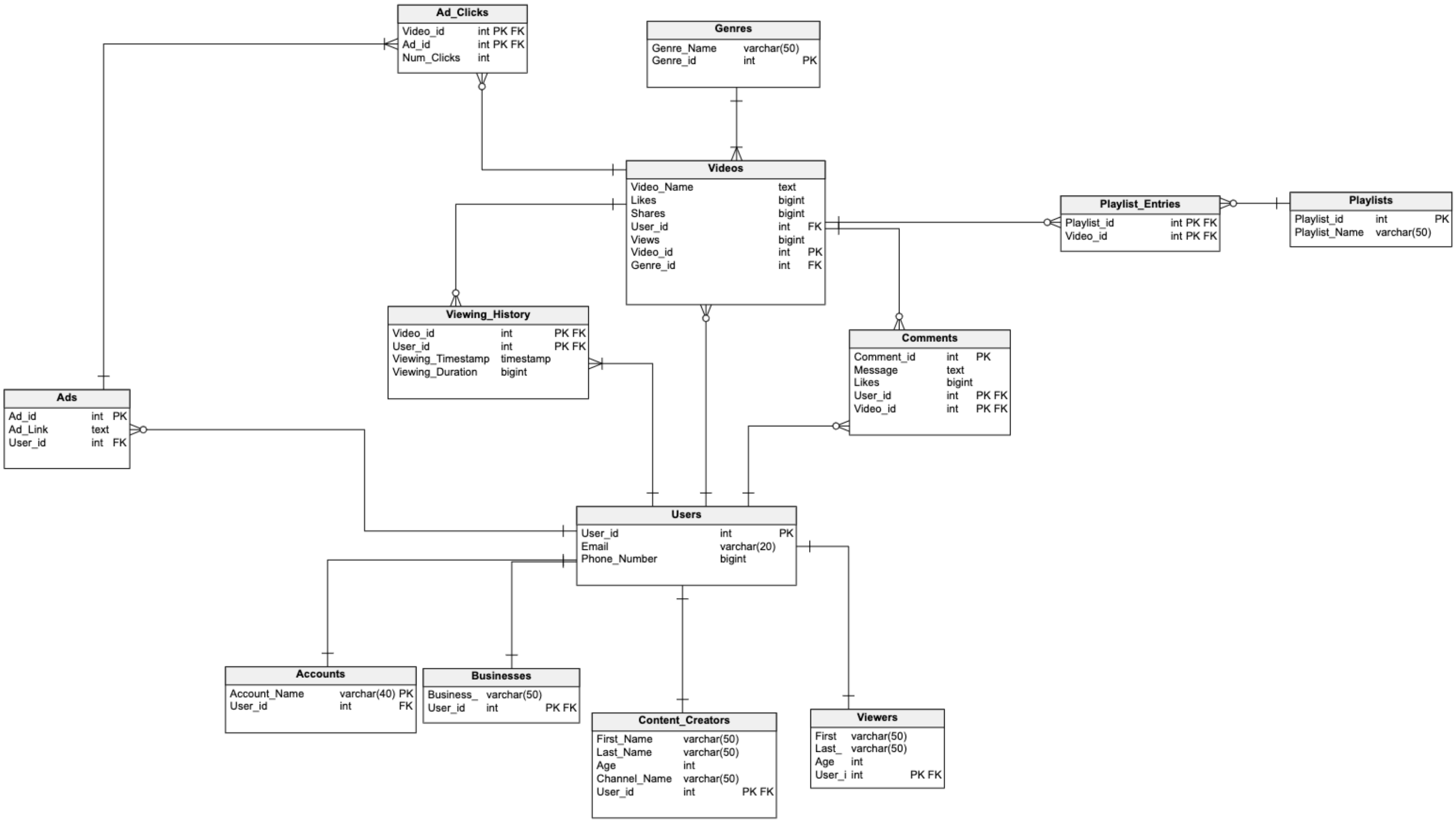

Figure 2: This is a snapshot of the converted final relational model into the physical model using Vertabelo.

Further Remarks

Now that we have constructed the physical model on Vertabelo, we will begin populating the tables with data and writing SQL queries based on our user stories. The assumptions listed above account for any anomalies, and we have written up explanations of each query on the python file. This model is by no means the full representation of YouTube's database, as construction of the entire model is beyond the scope for this project. However, our updated assumptions and final model account for our user stories and overall objective for the project.