Youtube Database

Ayush Gupta-2014CS50281
Bipul Kumar-2014CS50282

Due date: March 15, 2017, 11:55pm IST

1 Project Description

Our project involves building a simple video library just like youtube. This was motivated by the vastness of youtube, how much content it can store and deliver at any time, despite its largeness how quick it serves and whether we can successfully emulate at least a bare-bones Youtube video library.

The complete system allows users directly to query videos from the creater's library. Users can like, dislike, comment to a video and even subscribe to a channel. Users can view a playlist of videos. They can customize their search to get videos with specific property like videos with view greater than 10000 etc.

Our system keeps track of all videos and channels statistics and reveal them in real time.

The entities & attributes that were designed by us are shown in Table 1, while the ER Diagram corresponding to this is shown in Figure 1.

2 Data Sources and Statistics

We wrote our web crawlers in python to scrap data from
https://www.youtube.com/
using youtube apis
https://www.googleapis.com/youtube/v3/channels
https://www.googleapis.com/youtube/v3/playlists
https://www.googleapis.com/youtube/v3/playlistItems
https://www.googleapis.com/youtube/v3/videos
https://www.googleapis.com/youtube/v3/commentThreads

This resulted into 4 tables, channelList, playlistList, videoList, commentList. The code for crawlers is included in code section. We cleaned this data using our c++ scripts to change the format of date and time into timestamp datatype of postgres. We also converted duration into seconds using python script. The code for the script is included in code section.

Initially the 4 raw tables had lots of redundancies. So we applied 1NF, 2NF, 3NF and BCNF rules. This resulted into 7 tables. Table schema and related information is included below.

ChannelList.csv
The column names of the csv file are:

Table 1: **List of Entities and Attributes**

Entity	**Attributes**
videoid_info	videoid,videotitle,channelid,publishedat,defaultaudiolanguage,duration, caption,viewcount,likecount,dislikecount,commentcount
channelid_playlistid	playlistid,channelid
channelid_channeltitle	channelid,channeltitle
channelid_info	channelid,publishedat,country,subscribercount,videocount,viewcount,commentcount
playlist_info	playlistid,playlisttitle,channelid,playlistvideocount
playlistid_videoid	videoid,playlistid
commentid_info	commentid,videoid,channelid,likeCount,publishedat
email_password	email,password,permissions
email_videoid_comment	email,videoid,comment

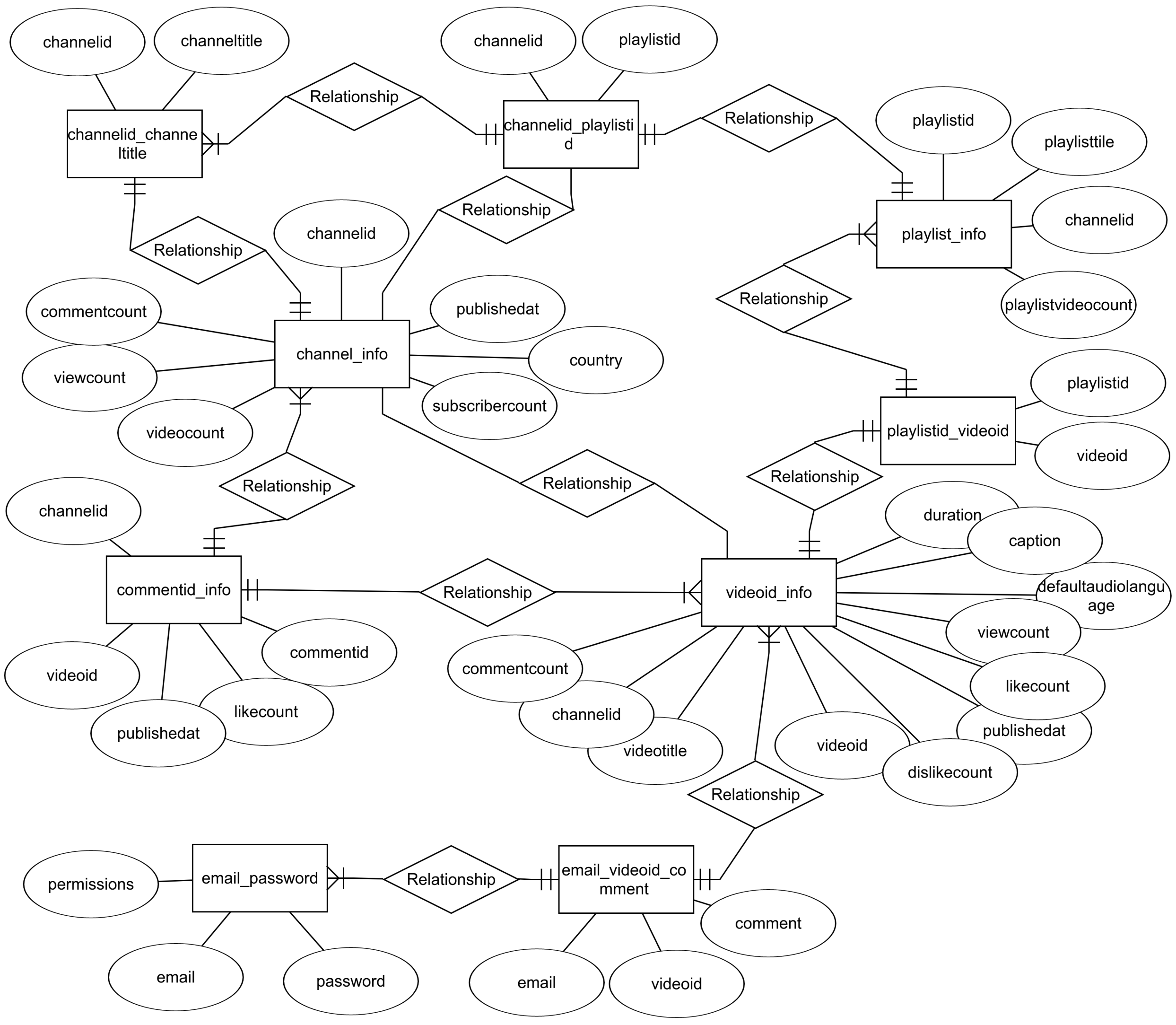

Figure 1: ER diagram

1. channelId
2. channelTitle
3. publishedAt
4. country
5. subscriberCount
6. videoCount
7. viewCount
8. commentCount
9. playlistId

1. 1NF: It is already done due to diff row of playlist and also because each element of each row is atomic.
2. 2NF: No split because Candidate key is playlistID alone, which uniquely determines a row.

Table 2: **Data type of attributes**

Attribute Name	**Data Type**
videoid,videotitle,channelid,playlistid, channeltitle,playlisttitle,commentid,email,password, comment	varchar
viewcount,playlistvideocount	bigint
publishedAt	timestamp
defaultaudiolanguage,country	char(10)
duration,likecount,dislikecount,commentcount,subscribercount,videocount,permissions	int
caption	boolean

3. 3NF: Split Table1 because Non-prime Channel Title or ChannelId determined other columns.
   Channel Title and ChannelId also need to be kept separate as one of these will determine others.
   Table1: channelId, playlistID
   Table2: channelId, channelTitle
   Table3: channelId, publishedAt,country, subscriberCount,videoCount

4. BCNF: It is satisfied because for each functional dependency, all the left terms are super keys in all the 3 tables.

PlaylistList.csv
The column names of the csv file are:

1. playlistId
2. playlistTitle
3. playlistChannelId
4. playlistVideoCount
5. videoId

1. 1NF: It is satisfied because each element of each row is atomic.
2. 2NF: VideoId is the candidate key and it uniquely identifies a row.
3. 3NF: PlaylistID which is a non-prime attribute determines all other non-prime attributes
   So we split the table into two tables:
   Table1: playlistId videoId
   Table2: playlistId playlistTitle playlistChannelId playlistVideoCount
   playlistid_videoid
4. BCNF: It is satisfied because for each functional dependency, all the left terms are super keys in all the 3 tables.

VideoList.csv
The column names of the csv file are:

1. videoId
2. videoTitle
3. channelId
4. publishedAt
5. defaultAudioLanguage
6. duration
7. caption
8. viewCount

Table 3: **Statistics**

Table Name	**No. of Tuples**	**Time to load(ms)**	**Size of raw data**	**Size(raw data) post cleanup**
videoid_info	4108	127.746	1.5MB	726KB
channelid_playlistid	3843	81.877	233.9KB	229.4KB
channelid_channeltitle	1661	30.885	65.5KB	65.5KB
channelid_info	129	20.648	10.4KB	9.7KB
playlist_info	1148	36.268	89.1KB	89.1KB
playlistid_videoid	4208	80.941	485.1KB	423.5KB
commentid_info	1621	28.266	197KB	155.8
email_password	1	-	-	-
email_videoid_comment	1	-	-	-

9. likeCount

10. dislikeCount

11. commentCount

1. 1NF: It is satisfied because each element of each row is atomic.

2. 2NF: VideoID is the only candidate key and it uniquely identifies a row.

3. 3NF: It is satisfied because no non-prime attribute can determine any other column.

4. BCNF: It is satisfied because for each functional dependency, all the left terms are super keys in all the 3 tables as each superkey needs to have videoID.

CommentList
The column names of the csv file are:

1. commentId

2. videoId

3. authorDisplayName

4. authorChannelId

5. likeCount

6. publishedAt

1. 1NF: It is satisfied because each element of each row is atomic.

2. 2NF: commentID is the only candidate key and it uniquely identifies a row.

3. 3NF: authorChannelId which is non-prime can uniquely determine authorDisplayName. So split the table:
   Table1: commentId videoId authorChannelId likeCount publishedAt Table2: authorChannelId authorDisplayName

4. BCNF: It is satisfied because for each functional dependency, all the left terms are super keys in all the 3 tables as each superkey needs to have commentID.

3 Functionality and Working

1. User's view of the system

   (a) Home:
   The front page consists of two options: register and login. If someone wants to create a new account he will click on register to signup, which if someone already has a account he will click on login button.

   (b) Register:
   The register page asks user to enter email, password and permission(u,a). It checks for format of email. we have unique constraint with email field so already in use email will not work and it will ask user to enter different email id. If permission is "0", then its a regular user, "1" means the person is moderator. On successful register, user is sent back to Home page.
   On registering, a insert query is sent to email_password table to insert user details. Password is MD5 hashed and stored in table.

   (c) Login:
   On login page user has to enter registered email and password. On pressing login button password is MD5 hashed and a query to verify email and hashed password is sent to the database. On successful login user is sent to Homepage and a Session variable. The database also checks the permission value which may be 0 or 1 and then saves the email and permission value in the Session variable.

   (d) Homepage:
   For a logged in user, this page shows three options: "Search Video", "Search Playlist" and "Search Channel". Clicking on the link redirects user to one of the pages from (Video Search Page, Playlist Search Page, Channel Search Page).

   (e) Video Search Page:
   This page contains a list of boxes where user can make any combination of a query by twiking the boxes. On submitting a corresponding query is fired to the database and the list of videos are shown. Each of the videos are clickable which takes user to Video Page. On clicking the video a query is sent to database to get all fields of the video.

   (f) Video Page:
   This page shows all information about a video. User can even comment, like and dislike the video. On submitting a new comment, the comment table is updated correspondingly. Also commentcount attribute of videoid_info table is increased on submitting the comment.

   (g) Playlist Search Page:
   This page contains a list of boxes where user can make any combination of a query by twiking the boxes. On submitting a corresponding query is fired to the database and the list of playlists are shown. Each of the playlists are clickable which takes user to playlist page. On clicking the playlist a query is sent to database to get all fields of the playlist which is displayed in playlist page.

   (h) Playlist Page:
   This page shows all information about a playlist like its title, number of videos, name of channel where it belongs etc. This page also shows list of videos which belongs to this playlist. Clicking on a particular video will take the user to a corresponding video page which contain all details regarding to the video.

   (i) Channel Search Page:
   This page contains a list of boxes where user can make any combination of a query by twiking the boxes. On submitting a corresponding query is fired to the database and the list of channels are shown. Each of the channels are clickable which takes user to Channel Page. On clicking the channel a query is sent to database to get all fields of the channel which is then shown in Channel Page.

   (j) Channel Page:
   This page shows all information about a channel like channeltitle, channeld, publishing date, country, subscribercount, videocount, viewcount, commentcount. Channel page also lists all of its playlists. Clicking on a playlist takes the user to playlist page which contains all of the playlist's details and the list of videos in the playlist.

(k) If the user is moderator then in the Video Page then he also gets the ability to modify any video title. On submitting, a query is sent to the database to change the title of the video. He can also comment on the video which is again inserted in email_videoid_comment table in the database and the comment count for the video is increased by 1.

(l) A user is identified by server using the session data which is created at the time of his login. So deleting the session data logs him out. All the webpages after login page will have a logout button at the top. Clicking the button will erase the session data from server and send the user to Home page.

2. Special Functionality

(a) Constraints:
Every table has primary key constraint, which inherently includes not null and unique value constraints. Apart from the primary key constraint, we have referential constraint and foreign key constraint on videoid attribute in videoid_info & email_videoid_comment tables. videoid is primary key in videoid_info table while it is foreign key in email_videoid_comment table.

(b) Indexing:
We were querying using videoid a lot our queries. And also videoid table is one of the largest table in our database. So for faster and efficient searching in our database, we indexed the table on videoid using B-Tree indexing. For this we used the below command:
CREATE INDEX videoid_index ON videoid_info (videoid);

(c) Views:
We created a materialised view playlist_info_channel which is join on channelid between channelid_channeltitle and playlist_info, to easily access and display channeltitle along with playlist-info.

Create materialized view playlist_info_channel as Select playlist_info.*, channelid_channeltitle.channeltitle from channelid_channeltitle JOIN playlist_info on channelid_channeltitle.channelid=playlist_info.channelid
We also created materialized view playlist_videos_info.

Create materialized view playlist_videos_info as Select videoid_info.*,playlistid_videoid.playlistid From playlistid_videoid JOIN videoid_info On playlistid_videoid.videoid=videoid_info.videoid;

(d) Access Privileges:
We have implemented two user mode: normal and moderator. Normal: This type of users can only search and view data. They can also comment on a video, which on submission, gets updated in videoid_info table immediately.
Moderator: This user has all the permissions of a normal user. Apart from that he can also modify a videotitle which will be updated on submission immediately.

(e) Triggers:

```
create or replace function refresh_mat_view()
returns trigger language plpgsql
as $$
begin
refresh materialized view playlist_videos_info;
return null;
end $$;

create trigger refresh_mat_view
after update
on videoid_info for each statement
execute procedure refresh_mat_view();
```

We created this trigger to update the materialized view "playlist_videos_info" whenever a moderator changes a video title on Video_Page.
The trigger basically keeps the View updated on "UPDATES" to videoid_info Table.
As we don't provide functionality to update other things, hence the other materialized views don't need a trigger now.
Normalising our tables have saved us from making so many other triggers which could have been needed for resolving updation anomalies.

3. List of Queries:

   (a) Register Page:
      i. SELECT * from email_password WHERE email='$email'
      ii. INSERT INTO email_password VALUES('$email','$hashed_password','$permission')

   (b) Login:
      i. SELECT * from email_password WHERE email='$email' AND password='$hashed_password'

   (c) Video Search Page:
      i. SELECT videoid,videotitle,viewcount FROM videoid_info WHERE viewcount >= 0 AND viewcount<1300 AND viewcount>1400 AND duration>1100 AND dislikecount>800 AND likecount>500 AND likecount=600 AND commentcount>200 ORDER BY viewcount
      ii. SELECT videoid,videotitle,viewcount FROM videoid_info WHERE viewcount>=0 ORDER BY viewcount DESC
      iii. SELECT videoid,videotitle,viewcount FROM videoid_info WHERE viewcount>=0 AND viewcount>200 ORDER BY viewcount DESC
      iv. SELECT videoid,videotitle,viewcount FROM videoid_info WHERE viewcount>=0 AND viewcount>200 AND duration<300 ORDER BY viewcount DESC

   (d) Video Page:
      i. SELECT * from videoid_info WHERE videoid='$videoid'
      ii. SELECT * from channelid_channeltitle WHERE channelid='$row[channelid]'
      iii. SELECT * from playlistid_videoid WHERE videoid='$row[videoid]'
      iv. SELECT * from playlist_info WHERE playlistid='$row2[playlistid]'
      v. SELECT * from email_videoid_comment WHERE videoid='$row[videoid]'
      vi. INSERT INTO email_videoid_comment VALUES ('$email_comment','$row[videoid]','$comment_text')
      vii. UPDATE videoid_info SET commentcount='$new_comment_count' WHERE videoid='$row[videoid]'
      viii. UPDATE videoid_info SET videotitle='$new_title' WHERE videoid='$row[videoid]'

   (e) Playlist Search Page:
      i. Select * from playlist_info where playlistvideocount>10 AND playlistvideocount<200 ORDER BY playlistvideocount DESC
      ii. Select * from playlist_info where playlisttitle=Vlog ORDER BY playlistvideocount
      iii. Select * from playlist_info_channel WHERE channeltitle=$channeltitlesearched ORDER BY plalistvideocount DESC

   (f) Playlist Page:
      i. Select * from playlist_info_channel;
      ii. select * from playlist_videos_info where playlist_videos_info.playlistid= $playlistid_current_page

   (g) Channel Search Page:
      i. Select * from channelid_info where subscribercount>1000 AND videocount>1000 AND viewcount>10000 AND commentcount>2000 AND country=US ORDER BY subscribercount
      ii. Select * from channelid_info where viewcount<9090 AND commentcount>4200 ORDER BY videocount DESC

   (h) Channel Page:
      i. Select * from channelid_info where channelid=$channelid_current;
      ii. Select * from channelid_channeltitle where channelid=$channelid_current;
      iii. select * from playlist_info_channel where channelid=$channelid_current;

   (a) Query Running Time

Table 4: **Query Running Time**

Query Number	Average Running Time
(a) i	0.229 ms
(a) ii	21.206 ms
(b) i	0.261 ms
(c) i	1.101 ms
(c) ii	7.682 ms
(c) iii	4.027 ms
(c) iv	3.663 ms
(d) i	0.372 ms
(d) ii	0.278 ms
(d) iii	0.222 ms
(d) iv	0.288 ms
(d) v	0.290 ms
(d) vi	22.448 ms
(d) vii	18.027 ms
(d) viii	24.749 ms
(e) i	0.455 ms
(e) ii	0.564 ms
(e) iii	0.410 ms
(f) i	0.400 ms
(f) ii	1.958 ms
(g) i	0.379 ms
(g) ii	0.339 ms
(h) i	0.220 ms
(h) ii	0.252 ms
(h) iii	0.306 ms

4 Codes

4.1 Get info about Channel

```
channelUrl = "https://www.googleapis.com/youtube/v3/channels"
channelParam = {
        "part": "id,snippet,statistics,contentOwnerDetails",
        "key": "AIzaSyAIeT0DZSZPNgDlaS_3NsaOiMCmQcfI0kb"
}
for channel in channelList:
        channelParam["id"] = channel
        channelPage = requests.request(
        method="get", url=channelUrl, params=channelParam)
        channelPageJson = json.loads(channelPage.text)
        channelId = channelPageJson["items"][0]["id"]
        channelTitle = (channelPageJson["items"][0]["snippet"]["title"]
        ).encode(encoding='UTF-8')
        publishedAt = channelPageJson["items"][0]["snippet"]["publishedAt"]
        if "country" not in channelPageJson["items"][0]["snippet"]:
                country = ""
        else:
                country = (channelPageJson["items"][0]["snippet"]["country"]
                ).encode(encoding='UTF-8')
        subscriberCount =
        channelPageJson["items"][0]["statistics"]["subscriberCount"]
        videoCount = channelPageJson["items"][0]["statistics"]["videoCount"]
        viewCount = channelPageJson["items"][0]["statistics"]["viewCount"]
```

```
            commentCount = channelPageJson["items"][0]["statistics"]["commentCount"]
            playlistUrl = "https://www.googleapis.com/youtube/v3/playlists"
            playlistParam = {
                    "part": "snippet,contentDetails",
                    "key": "AIzaSyAIeT0DZSZPNgDlaS_3NsaOiMCmQcfI0kc",
                    "maxResults": 50
            }
            playlistParam["channelId"] = channel
            playlistPage = requests.request(
            method="get", url=playlistUrl, params=playlistParam)
            playlistPageJson = json.loads(playlistPage.text)
            for item in playlistPageJson["items"]:
                    playlistId = item["id"]
                    channelCSVWriter.writerow([
                    channelId,channelTitle,publishedAt,country,subscriberCount,
                    videoCount,viewCount,commentCount,playlistId
                    ])
    channelCSV.close()
```

4.2 Get info about Playlist

```
    videoList = []
    for channel in channelList:

            playlistUrl = "https://www.googleapis.com/youtube/v3/playlists"
    playlistParam = {
            "part": "snippet,contentDetails",
            "key": "AIzaSyAIeT0DZSZPNgDlaS_3NsaOiMCmQcfI0kb",
            "maxResults": 50
    }
    playlistParam["channelId"] = channel
    playlistPage =
    requests.request(method="get", url=playlistUrl, params=playlistParam)
    playlistPageJson = json.loads(playlistPage.text)
    for item in playlistPageJson["items"]:
            playlistId = item["id"]
            playlistTitle = (
            item["snippet"]["title"]).encode(encoding='UTF-8')
            playlistChannelId = item["snippet"]["channelId"]
            playlistVideoCount = item["contentDetails"]["itemCount"]

    playlistItemUrl = "https://www.googleapis.com/youtube/v3/playlistItems"
    playlistItemParam = {
            "part": "contentDetails",
            "key": "AIzaSyAIeT0DZSZPNgDlaS_3NsaOiMCmQcfI0kc",
            "maxResults": 50
    }
    playlistItemParam["playlistId"] = playlistId
    playlistItemPage =
    requests.request(method="get", url=playlistItemUrl, params=playlistItemParam)
    playlistItemPageJson = json.loads(playlistItemPage.text)
    for videoItem in playlistItemPageJson["items"]:
            videoId = videoItem["contentDetails"]["videoId"]
            videoList.append(videoId)
            playlistCSVWriter.writerow([
            playlistId,playlistTitle,playlistChannelId,
            playlistVideoCount,videoId])
```

4.3 Get info about Video

```
for video in videoList:
        videoUrl = "https://www.googleapis.com/youtube/v3/videos"
        videoParam = {
        "part": "snippet,statistics,contentDetails",
        "key": "AIzaSyAIeT0DZSZPNgDlaS_3NsaOiMCmQcfI0kb",
        "maxResults": 50
        }
        videoParam["id"] = video
        videoPage =
        requests.request(method="get", url=videoUrl, params=videoParam)
        videoPageJson = json.loads(videoPage.text)
        if len(videoPageJson["items"]) == 0:
                continue
        videoId = videoPageJson["items"][0]["id"]
        videoTitle = (
                videoPageJson["items"][0]["snippet"]["title"]
                ).encode(encoding='UTF-8')
        channelId = videoPageJson["items"][0]["snippet"]["channelId"]
        publishedAt = videoPageJson["items"][0]["snippet"]["publishedAt"]
        if "defaultAudioLanguage" not in videoPageJson["items"][0]["snippet"]:
                defaultAudioLanguage = "en"
        else:
                defaultAudioLanguage = videoPageJson["items"][0]["snippet"]
                ["defaultAudioLanguage"]
        duration = videoPageJson["items"][0]["contentDetails"]["duration"]
        caption = videoPageJson["items"][0]["contentDetails"]["caption"]
        viewCount = videoPageJson["items"][0]["statistics"]["viewCount"]
        if 'likeCount' not in videoPageJson["items"][0]["statistics"]:
                likeCount = 0
        else:
                likeCount = videoPageJson["items"][0]["statistics"]["likeCount"]
        if 'dislikeCount' not in videoPageJson["items"][0]["statistics"]:
                dislikeCount = 0
        else:
        dislikeCount = videoPageJson["items"][0]["statistics"]["dislikeCount"]
        if 'commentCount' not in videoPageJson["items"][0]["statistics"]:
                commentCount = 0
        else:
        commentCount = videoPageJson["items"][0]["statistics"]["commentCount"]

        videoCSVWriter.writerow([
        videoId,videoTitle,channelId,publishedAt,defaultAudioLanguage,
        duration,caption,viewCount,likeCount,dislikeCount,commentCount
        ])
```

4.4 Get info about Comment

```
commentUrl = "https://www.googleapis.com/youtube/v3/commentThreads"
commentParam = {
        "part": "snippet",
        "key": "AIzaSyAIeT0DZSZPNgDlaS_3NsaOiMCmQcfI0kb",
        "maxResults": 50
}
j = 0
for video in videoList:
        if j > 1000:
```

```
                        break
                j+=1
        commentParam["videoId"] = video
        commentPage = requests.request(
        method="get", url=commentUrl, params=commentParam
        )
        commentPageJson = json.loads(commentPage.text)
        i = 0
        if "items" not in commentPageJson:
                continue
        if (len(commentPageJson["items"]) == 0):
                continue
        for commentItem in commentPageJson["items"]:
                if i > 1:
                        break
                i+=1
                commentId = commentItem["id"]
                videoId = commentItem["snippet"]["videoId"]
                authorDisplayName = (
                commentItem["snippet"]["topLevelComment"]["snippet"]
                ["authorDisplayName"]
                ).encode(encoding='UTF-8')
                authorChannelId = commentItem["snippet"]["topLevelComment"]
                ["snippet"]["authorChannelId"]["value"]
                likeCount = commentItem["snippet"]["topLevelComment"]
                ["snippet"]["likeCount"]
                publishedAt = commentItem["snippet"]["topLevelComment"]
                ["snippet"]["publishedAt"]
                commentCSVWriter.writerow([
                commentId,videoId,authorDisplayName,authorChannelId,
                likeCount,publishedAt
                ])
```

4.5 Cleanup Data

- Timestamp

```
vector<string> v;
void read_input()
{
        ifstream infile;
        infile.open("commentid_info_column_ahead.csv");
        string dummy;
        while(getline(infile,dummy))
        {
                v.push_back(dummy);
        }
        infile.close();
}

void print_output()
{
        ofstream outfile;
        outfile.open("commentid_info_column_ahead_modified.csv");
        for(int j=0;j<int(v.size());j++)
        {
                outfile<<v[j]<<endl;
        }
        outfile.close();
```

```
}


int main()
{
        read_input();
        cout<<"Size is"<<int(v.size())<<endl;
        for(int i=1;i<int(v.size());i++)
        {
                cout<<"for"<<i<<endl;
                int first_comma=0;
                string dummy="";
                for(int j=0;j<int(v[i].length());j++)
                {
                        if(v[i].at(j)==',')
                        {
                                first_comma=j;
                                dummy+=v[i].at(j);
                                break;
                        }
                        else
                        {
                                dummy+=v[i].at(j);
                        }
                }

                for(int j=(first_comma+1);j<int(v[i].length());j++)
                {
                        if(v[i].at(j)=='T')
                        {
                                dummy+=" ";
                                first_comma=j;
                                break;
                        }
                        else
                        {
                                dummy+=v[i].at(j);
                        }
                }
                for(int j=(first_comma+1);j<int(v[i].length());j++)
                {
                        if(v[i].at(j)=='.')
                        {
                                first_comma=j;
                                break;
                        }
                        else
                        {
                                dummy+=v[i].at(j);
                }
        }
        first_comma+=4;
        for(int j=(first_comma+1);j<int(v[i].length());j++)
        {
                dummy+=v[i].at(j);
        }
        v[i]=dummy;
}
```

```
print_output();
return 0;
}
```

- Duration

```
import isodate
fo = open("file_to_be_modified.csv")
a = 0
b = []
z = 1
for line in fo:
        if z ==1:
                z+=1
                continue
        temp = ""
        for i in xrange(len(line)):
                if line[i] != ",":
                        temp += line[i]
                else:
                        a = i
                        break
        dur=isodate.parse_duration(temp)
        dur_in_sec = dur.total_seconds()
        line = str(dur_in_sec) + line[a:]
        b.append(line)
fo.close()


fw = open("file_to_be_modified.csv", "w")

for line in b:
        fw.write(str(b))
fw.close()
```

End.