COMP 375 Final Project: Reverse-Engineering YouTube's Database

Kenny Callaghan
Ali Erkan
Fall 2019

Goal

The purpose of this project is to attempt to successfully reverse-engineer YouTube into a PostgreSQL database. Using only resources available to a front-end consumer, I've created a model for how I believe YouTube's internal database looks, or how it may have looked at one time. I've tried to be as exhaustive in my model as possible, and have also implored the concepts we learned in class about normalization to the creation of the tables in this database. The end result is something that I am content with.

Introduction

YouTube is the biggest video-sharing platform on the internet, and it has been for years. With billions of users globally spending collective years watching videos on a daily basis, the system must be very developed and robust in order to meet the large and growing needs of its astronomically large user-base.

This project serves as my attempt to reverse-engineer the database behind YouTube. As someone who has used the service extensively, from very close to its creation until now, I felt comfortable in undertaking this task. In this project, I have combined my own prior knowledge of the service and its perceived inner machinations with my own recent education on the principles of good database design to create something that I feel could accurately depict YouTube's back-end to a novice.

Normalization

Throughout this course, we learned a number of different classifications for normalizing data in a tabular database. Each higher level of normalization is more rigorous than the previous, and includes all the aspects of the previous normal forms. In the initial stages of development of my project, the schema hovered somewhere around Second and Third Normal Form. At this point, I would argue that this schema is in BCNF, because:

- The table has a primary key, and there are no multi-value columns.
- All non-key attributes are functionally depend on the key.
- There are no transitive dependencies between columns.
- For any functional dependency X->Y, X is a superkey.

Data

In order to simulate the load of actual data, I filled the tables in my database using Python programs that I wrote myself. For each table, I produced 100,000 rows of data. While this may be microscopic in terms of YouTube's scope, for a college student on an Ultrabook, this was very consuming in both time and processing resources. For URLs, I generated a random alphanumeric string of length 11, which serves as the portion after "v=" in a genuine YouTube URL. For titles, descriptions, and names, I downloaded a list of a few thousand random words, that were concatenated together to produce the data in the tables. Below is a screenshot of the program that was used to fill the Channels table.

Performance

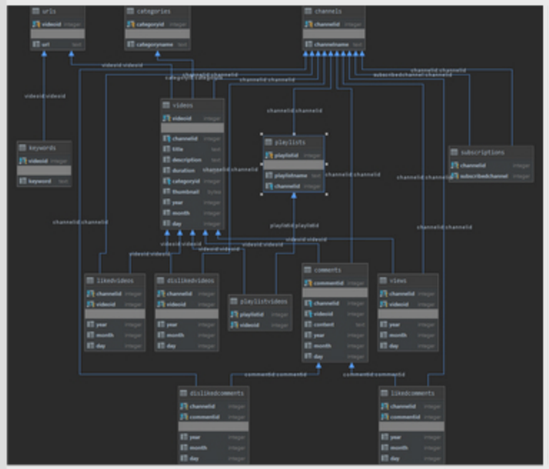
Populating the tables was a long and tedious process. Although the programs were fairly basic, the actual time it took for all of them to execute in their entirety was close to two days. Datagrip ran out of memory and crashed multiple times throughout.

Once the tables were populated, a number of queries were performed and the time it took for the processor to execute each query was recorded. Overall, most complex queries similar to the ones that would be used by YouTube developers themself were executed in between 50 and 350 milliseconds.

One aspect of SQL that was tested was the performance increase created by indexing certain columns. For this test, I ran the same query using one column with indexing, and one column without. The result, perhaps unsurprising, is that the indexed column completed with over a 100% increase in performance compared to its unindexed counterpart.

Schema

On the right, I have attached a visualization of my database schema that I have produced using Datagrip. After my assessment of the service, I decided that the schema should consist of 14 tables, with 57 columns in total. Most tables are related to each other using an ID column (usually video, but also occasionally channel, comment, and playlist). This schema underwent a number of revisions throughout the course of this project, before this ultimate version was finally established. I believe this schema to be in Boyce-Codd Normal Form (more on that under "Normalization").




```
f = open("Words.txt", "r")

for i in range(3251):
    #print(f.readline())
    word = f.readline()
    words.append(word[0:len(word)-1])

def generateChannelName():
    name = random.choice(words) + random.choice(words)
    return name

def insert_channel(channel_name):
    """ insert a new channel into the channels table """
    sql = """INSERT INTO Channels (channelId, channelName) VALUES(%s,%s) RETURNING channelId;"""
    conn = None
    channelId = None
    try:
        # read database configuration
        params = config()
        # connect to the PostgreSQL database
        conn = psycopg2.connect(**params)
        # create a new cursor
        cur = conn.cursor()
        # execute the INSERT statement
        cur.execute(sql, (channel_name,))
        # get the generated id back
        channelId = cur.fetchone()[0]
        # commit the changes to the database
        conn.commit()
        # close communication with the database
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()

    return channelId

for i in range(100000):
    name = generateChannelName()
    channel_to_insert = (i, name)
    insert_channel(channel_to_insert)

print("Done!")
```

Conclusion

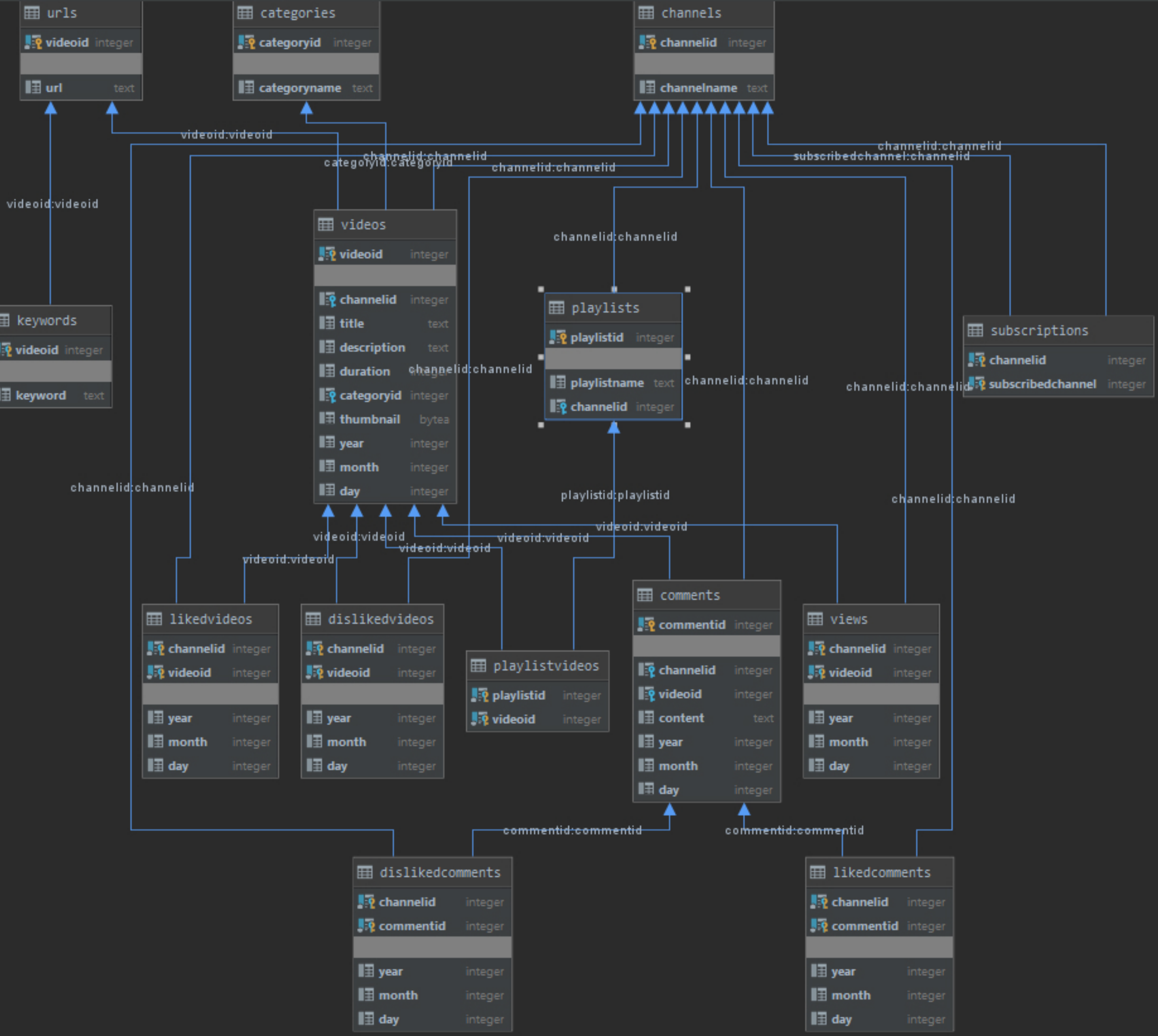
In conclusion, I feel as this project was very worthwhile towards my understanding of SQL, databases, and really cemented what I learned in this course as a whole. I am surprised myself how much I enjoyed the material we learned, and I am looking forward to working with large amounts of data at some point again in the future. I am left satisfied with my knowledge and work that I produced throughout this course.

AWS Information

- host=database-1.cf4dgpl9kyvw.us-east-2.rds.amazonaws.com
- database=youtubedb
- user=postgres
- password=changeme

Contact

Kenny Callaghan
Ithaca College
Email: kcallaghan@ithaca.edu


urls
videoid integer
url text
categories
categoryid integer
categoryname text
channels
channelid integer
channelname text
videoid:videoid
channelid:channelid
categoryid:categoryid
channelid:channelid
channelid:channelid
subscribedchannel:channelid
videoid:videoid
videos
videoid integer
channelid integer
title text
description text
duration integer
categoryid integer
thumbnail bytea
year integer
month integer
day integer
channelid:channelid
keywords
videoid integer
keyword text
playlists
playlistid integer
playlistname text
channelid integer
channelid:channelid
channelid:channelid
subscriptions
channelid integer
subscribedchannel integer
channelid:channelid
playlistid:playlistid
channelid:channelid
videoid:videoid
videoid:videoid
videoid:videoid
videoid:videoid
videoid:videoid
likedvideos
channelid integer
videoid integer
year integer
month integer
day integer
dislikedvideos
channelid integer
videoid integer
year integer
month integer
day integer
playlistvideos
playlistid integer
videoid integer
comments
commentid integer
channelid integer
videoid integer
content text
year integer
month integer
day integer
views
channelid integer
videoid integer
year integer
month integer
day integer
commentid:commentid
commentid:commentid
dislikedcomments
channelid integer
commentid integer
year integer
month integer
day integer
likedcomments
channelid integer
commentid integer
year integer
month integer
day integer

```
f = open("Words.txt", "r")

for i in range(3251):
    #print(f.readline())
    word = f.readline()
    words.append(word[0:len(word)-1])

def generateChannelName():
    name = random.choice(words) + random.choice(words)
    return name

def insert_channel(channel_name):
    """ insert a new channel into the channels table """
    sql = """INSERT INTO Channels (channelId, channelName) VALUES(%s,%s) RETURNING channelId;"""
    conn = None
    channelId = None
    try:
        # read database configuration
        params = config()
        # connect to the PostgreSQL database
        conn = psycopg2.connect(**params)
        # create a new cursor
        cur = conn.cursor()
        # execute the INSERT statement
        cur.execute(sql, (channel_name),)
        # get the generated id back
        channelId = cur.fetchone()[0]
        # commit the changes to the database
        conn.commit()
        # close communication with the database
        cur.close()
    except (Exception, psycopg2.DatabaseError) as error:
        print(error)
    finally:
        if conn is not None:
            conn.close()

    return channelId

for i in range(100000):
    name = generateChannelName()
    channel_to_insert = (i, name)
    insert_channel(channel_to_insert)


print("Done!")
```