

Universidad Autónoma de Sinaloa

Facultad de Informática Culiacán -
Facultad de Ciencias de la Tierra y el Espacio

Maestría en Ciencias de la Información



Fundamentos de Bases de Datos

Profesor: Dr. Inés Fernando Vega López

Alumno: Eliel Avilez Valenzuela

Trabajo: Construcción de base de datos Trending YouTube
Video Statistics

Culiacán, Sinaloa. Julio de 2023

Introducción

En este reporte se presenta el proceso para construir una base de datos a partir del conjunto de datos Trending Youtube Video Statistics, el cual consiste en 10 archivos CSV que almacenan los datos de los videos en tendencia (cada archivo pertenece a una región en particular) y 10 archivos JSON los cuales guardan la relación entre los id de categoría de video con la descripción de la categoría, al igual que con los CSV, cada archivo pertenece a una región en particular.

El objetivo es construir una tabla global que almacene todos los datos de todos los CSV, así como las descripciones de las categorías contenidas en los archivos JSON, también codificar 6 consultas indicadas para esta tarea, y medir su tiempo promedio de ejecución en frío y en caliente para cada una de ellas (30 ejecuciones para cada una).

Se utilizarán scripts de Python para extraer la relación entre categorías y sus id de los archivos JSON, para limpiar la información de los archivos CSV, y para calcular los tiempos promedio de ejecución. También, se utilizaron scripts en bash para almacenar los tiempos de ejecución de las consultas de manera automatizada, para después calcular sus promedios con código de Python.

1. Preparación de archivos CSV para cargar a base de datos

1.1. Creación de archivo CSV de categorías por región

En los archivos CSV que almacenan el conjunto de datos de videos por región, existe una columna llamada *category_id*, la cual contiene valores numéricos. En los archivos JSON de categorías por región, existe una relación entre el valor numérico de *category_id* con la descripción de la categoría de alguna región determinada, por lo tanto, es necesario extraer esta relación entre esos 2 atributos.

Para solventar esta necesidad, se pretende crear un archivo CSV que contenga el esquema $\langle region, category_id, category \rangle$; de esta manera se pueden cargar este archivo CSV a la base de datos como una nueva relacion, completando esto, será posible efectuar JOINs entre las tablas del conjunto de datos de videos, y así poder contar con la descripción de las categorías correspondientes para cada una de las regiones.

A continuación, se presenta un fragmento de un archivo JSON de categorías para ilustrar su estructura:

```
1  {
2    "kind": "youtube#videoCategoryListResponse",
3    "etag": "\"ld9biNPKjAjjV7EZ4EKeEGrhao/1v2mrzYSYG6onNLt2qTj13hkQZk\"",
4    "items": [
5      {
6        "kind": "youtube#videoCategory",
7        "etag": "\"ld9biNPKjAjjV7EZ4EKeEGrhao/Xy1mB4_yLrHy_BmKmpBgty2mZQ\"",
8        "id": "1",
9        "snippet": {
10          "channelId": "UCBR8-60-B28hp2BmDPdntcQ",
11          "title": "Film & Animation",
12          "assignable": true
13        }
14      }
15    .
16    .
17    .
18  }
```

Los elementos relevantes dentro de este fragmento, son el valor de *id* y el valor de *title*, los cuales corresponden a *category_id* y *category* respectivamente. Cabe señalar que los objetos que contienen los atributos relevantes, están dentro del arreglo *items*.

Por medio de un script en Python se obtendrán estos valores de cada archivo JSON y se generará un archivo CSV conteniendo estos datos de todas las regiones.

Primero importamos la biblioteca *os* para hacer manejo de archivos en directorios del sistema, *json* para manejar archivos JSON dentro del script, y *pandas* para hacer uso de DataFrames.

Definiendo la ruta que contiene todos los archivos, iteramos a través de todos los archivos que existen dentro de esa ruta y almacenamos los nombres de todos aquellos archivos que tienen la terminación “.json” en una lista llamada *archivosJSON*.

A continuación se presenta el código utilizado para realizar esta tarea:

```
1  import os
2  import json
3  import pandas as pd
4
5  # Guardar nombres de archivos JSON en lista
6  ruta = '/home/eliel/Posgrado/Semestre2/FundamentosBaseDeDatos/Tarea3/
7         archive/'
8  archivosJSON = [nombreArchivo for nombreArchivo in os.listdir(ruta)
9                  if nombreArchivo.endswith('.json')]
10
```

Posterior a esto, dentro de un ciclo *for*, iteramos a través de los valores de la lista *archivosJSON*, utilizando cada valor de la lista para abrir los archivos JSON uno a uno e insertar su contenido en un diccionario llamado *datosArchivo* en cada iteración (líneas 9-11). Dentro de cada archivo JSON existe un arreglo que lleva por nombre *items*, este será una lista que contiene objetos en forma de diccionario, cada uno de estos objetos contiene la relación entre id de categoría y su descripción correspondiente.

Ya que se tienen todos los datos en *datosArchivo*, entonces iteramos sobre el arreglo *datosArchivos["items"]* (línea 15) que como mencionamos anteriormente, contiene todos los objetos con la información que se busca obtener; al acceder a cada uno de estos objetos, almacenamos el id de categoría y su descripción en una lista llamada *datosDF* junto con el prefijo de la región a la que corresponde (esta se extrae del nombre del archivo JSON origen) e incrementamos el contador *i* una unidad (líneas 15-19), esto para llevar el control de las posiciones de los objetos a los que se accesan.

Después, se insertan los datos de *datosDF* en un DataFrame llamado *dfRegion* de tres columnas: *region*, *category_id* y *category*, la tercera columna almacenará la descripción de la categoría (líneas 22-23).

Una vez teniendo el DataFrame de la actual iteración listo, se concatena a un DataFrame que contendrá todos los datos de todas las regiones, este lleva por nombre *dfCompleto* (línea 26). Por último, el contenido de *dfCompleto* es vaciado en un archivo CSV que lleve por nombre “RegionsCategorys.csv” (línea 31).

A continuación, se presenta el código descrito anteriormente:

```
1 ##### Abrir archivos JSON y extraer sus datos #####
2 i = 0 # i: indice de categorias dentro de JSON
3
4 # Dataframe vacio que almacena todos los datos
5 dfCompleto = pd.DataFrame(columns = ['region', 'category_id', 'category'
6 ])
7
8 for archivo in archivosJSON:
9     # Abrir archivos JSON e insertar sus datos en datosArchivo
10    with open(ruta + archivo, 'r') as f:
11        datosArchivo = json.load(f)
12        f.close()
13
14    # Guardar en lista los valores de category_id y category
15    datosDF = []
16    for elemento in datosArchivo["items"]:
17        datosDF.append([archivo[0:2],
18                        datosArchivo["items"][i]["id"],
19                        datosArchivo["items"][i]["snippet"]["title"]])
20        i += 1
21
22    # Insertar datos de dataframe de region
23    dfRegion = pd.DataFrame(datosDF,
24                             columns=['region', 'category_id', 'category'])
25
26    # Agregar datos de dataframe de region en dataframe completo
27    dfCompleto = dfCompleto.append(dfRegion)
28
29    i = 0
30
31 # Guardar dataframe en CSV
32 dfCompleto.to_csv('RegionsCategorys.csv', index=False)
```

En la siguiente figura se puede apreciar la disposición de los datos de origen y destino de este proceso. La cantidad de archivos JSON origen es 10.

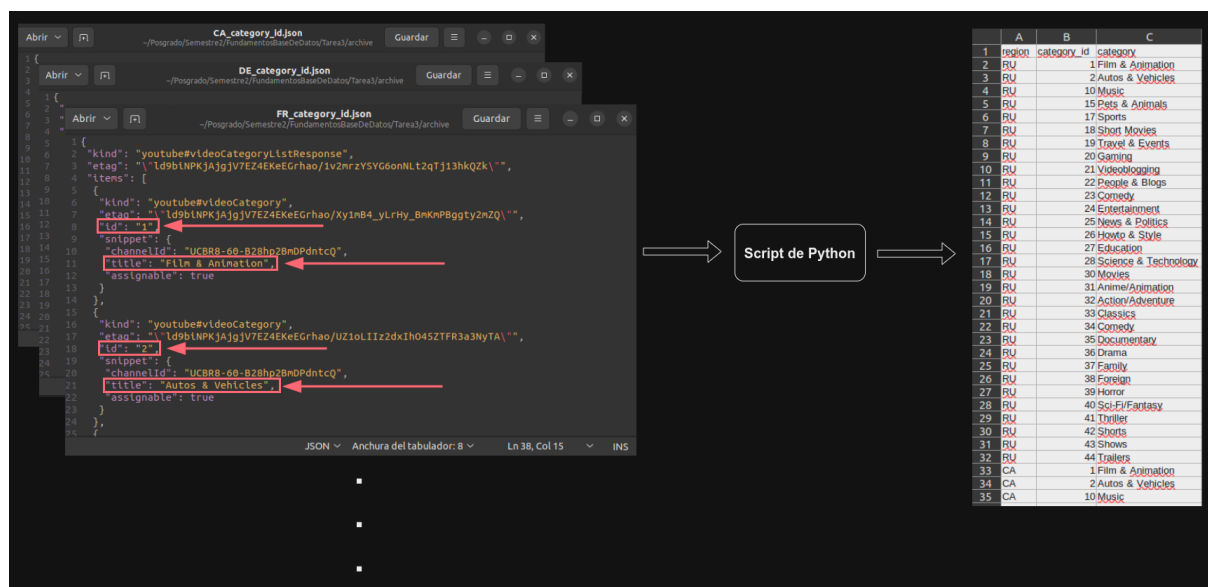


Figura 1.1: Archivos JSON origen (izquierda) y archivo CSV resultante (derecha).

1.2. Ajustes en archivos CSV de datos de videos

Los archivos que contienen los datos de videos de las distintas regiones requieren algunos ajustes antes de ser insertados a la base de datos.

La segunda columna de cada archivo es *trending_date*, la cual guarda una fecha con un formato *YY.DD.MM*, el cual preferentemente debe ser cambiado a *YYYY-MM-DD* para evitar problemas al insertar a una tabla de base de datos. Otro ajuste, es en la última columna llamada *description*, ya que esta columna almacena textos largos, puede contener comas en su texto, por lo que es importante que al inicio y al final de los valores de ese campo existan caracteres de comillas, de este modo el manejador de base de datos no va a interpretar estas comas intermedias como separadores del archivo CSV.

Con ayuda de un script de Python se harán estos ajustes, para después guardar la información limpia en nuevos archivos CSV.

Primero importamos la biblioteca *os* para hacer manejo de archivos en directorios del sistema, *csv* para manejar archivos CSV dentro del script, y *pandas* para hacer uso de DataFrames.

Definiendo la ruta que contiene todos los archivos, iteramos a través de todos los archivos

que existen dentro de esa ruta y almacenamos los nombres de todos aquellos archivos que tienen la terminación “.csv” en una lista llamada *archivosCSV*.

A continuación se presenta el código utilizado para realizar esta tarea:

```
1 import os
2 import csv
3 import pandas as pd
4
5 # Guardar nombres de archivos CSV en lista
6 ruta = '/home/eliel/Posgrado/Semestre2/FundamentosBaseDeDatos/Tarea3/
7         archive/'
8
9 archivosCSV = [nombreArchivo for nombreArchivo in os.listdir(ruta)
10                 if nombreArchivo.endswith( '.csv' )]
11
```

Posterior a esto, dentro de un ciclo *for*, iteramos a través de los valores de la lista *archivosCSV*, utilizando cada valor de la lista para abrir los archivos CSV uno a uno (líneas 5-6) e insertar su contenido en un DataFrame llamado *dfArchivo* en cada iteración. Una vez abierto el archivo, iteramos a través de él línea por línea con otro ciclo *for* (línea 11), la primera línea de cada archivo se guarda en una lista llamada *cabecero* para utilizarlo como el cabecero del DataFrame, el resto de las líneas se revisa si en la columna 15 (la última columna) es distinta de vacío y que no tenga comillas en sus extremos, si ambas condiciones se cumplen, entonces se agregan comillas al inicio y al final del valor de la columna para todas las líneas (líneas 19-20).

Así mismo, con la segunda columna se transforma la fecha que contiene el formato *YY.DD.MM* a *YYYY-MM-DD* (líneas 23-25).

Después se agrega la línea reparada a *listaLineasCSV* (línea 28) para posteriormente utilizar esta lista que contiene todas las filas del archivo CSV, como el contenido del DataFrame *dfArchivo* (línea 31); finalmente, insertamos todo el contenido del DataFrame a un nuevo archivo CSV que lleve por nombre el prefijo *CLEAN_* seguido del nombre del CSV origen (línea 34) e.g. *CLEAN_CAvideos.csv*.

A continuación, se presenta el código utilizado para realizar lo descrito:

```

1 ##### Abrir archivos CSV y extraer sus datos #####
2 i = 0
3
4 for nombreArchivo in archivosCSV:
5     with open(ruta + nombreArchivo, 'r', errors = 'replace') as archivo:
6         lectorCSV = csv.reader(archivo, delimiter = ',')
7
8         listaLineasCSV = []
9
10        # Leer linea por linea CSV
11        for linea in lectorCSV:
12            # Guardar cabecero del archivo
13            if(i == 0):
14                cabecero = linea
15                i += 1
16            else:
17                # Si la ultima columna no esta vacia y no esta
18                # entre comillas, agregar comillas
19                if(linea[15] != "" and linea[15][0] != '"'):
20                    linea[15] = '"' + linea[15] + '"'
21
22                # Dar formato YYYY-MM-DD a la fecha de la 2da columna
23                linea[1] = "20" + linea[1][0:2] +
24                    "-" + linea[1][6:8] +
25                    "-" + linea[1][3:5]
26
27                # Agregar linea arreglada a listaLineasCSV
28                listaLineasCSV.append(linea)
29
30        # Agregar datos de listaLineasCSV a un dataframe
31        dfArchivo = pd.DataFrame(listaLineasCSV, columns = cabecero)
32
33        # Guardar archivo CSV nuevo con informacion limpia
34        dfArchivo.to_csv('CLEAN_' + nombreArchivo, index = False)
35
36        i = 0
37

```


En la siguiente figura se puede apreciar de manera gráfica el cambio realizado con el script de Python a los archivos origen y el resultado en el destino. La cantidad de archivos CSV origen es 10.

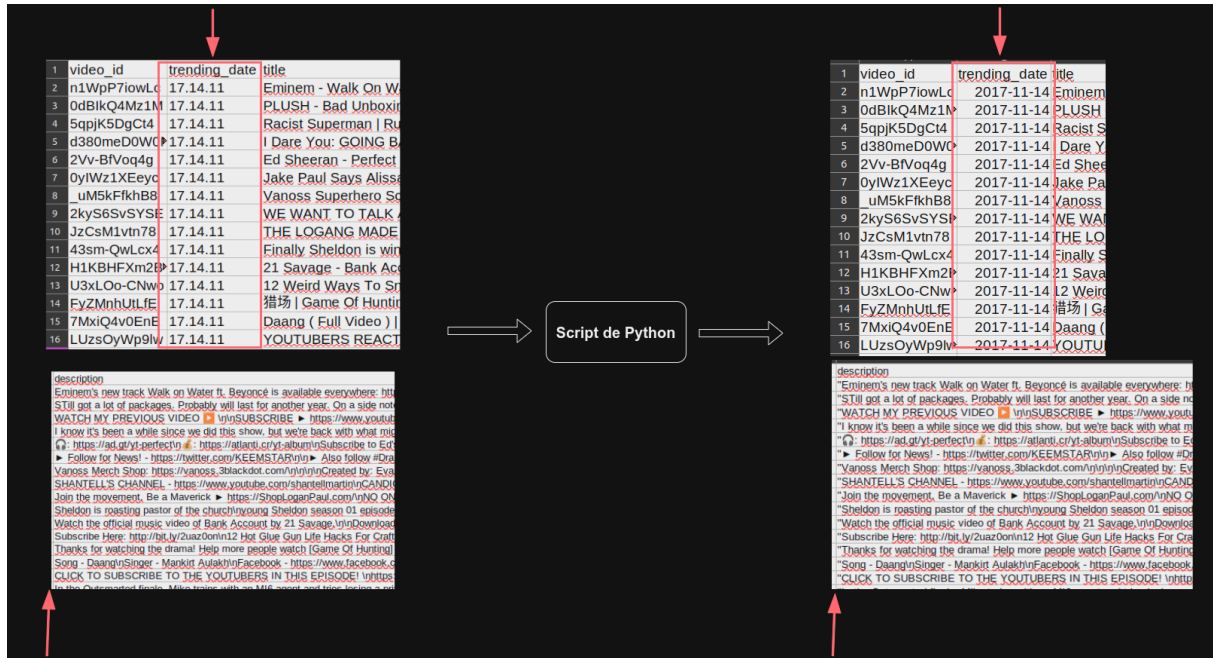


Figura 1.2: Archivos CSV origen (izquierda) y archivo CSV resultante (derecha).

2. Creación y carga de tablas en PostgreSQL

2.1. Creación de tablas

Para almacenar los datos de categoría de cada región, se creó la tabla *CategoryRegion*, y para guardar la información de los videos de cada región se crearon varias relaciones en la base de datos, un ejemplo es *DataVideos_Canada*, cada tabla para cada región llevará el nombre de su país correspondiente.

En el siguiente fragmento se presentan los scripts de creación para estas tablas:

```
1  — Categorías por region
2
3  CREATE TABLE CategoryRegion
4  (
5      Region CHAR(20)
6      ,Category_id INTEGER
7      ,Category CHAR(100)
8  );
9
10 — Videos Canada
11
12 CREATE TABLE DataVideos_Canada
13 (
14     video_id CHAR(20)
15     ,trending_date DATE
16     ,title CHAR(100)
17     ,channel_title CHAR(100)
18     ,category_id INTEGER
19     ,publish_time TIMESTAMP(2)
20     ,tags CHAR(1000)
21     ,views BIGINT
22     ,likes BIGINT
23     ,dislikes BIGINT
24     ,comment_count INTEGER
25     ,thumbnail_link CHAR(100)
26     ,comments_disabled BOOLEAN
27     ,ratings_disabled BOOLEAN
28     ,video_error_or_removed BOOLEAN
29     ,description CHAR(6000)
30 );
31 .
32 .
33 .
34
```

Los demás scripts de creación de tablas de datos para videos se omiten ya que su estructura es la misma que la de *DataVideos_Canada*.

Además de estas tablas, también se creó una tabla llamada *DataVideos_AllRegions* que albergará toda la información de todas las regiones, el esquema es muy similar a las demás tablas de DataVideos, solo que esta tendrá una columna llamada *region* para llevar el nombre de la región a la que pertenece la tupla, y el campo *category_id* será sustituido por *category*, que guardará la descripción de la categoría para cada tupla.

```
1 CREATE TABLE DataVideos_AllRegions
2 (
3     region CHAR(25)
4     ,video_id CHAR(20)
5     ,trending_date DATE
6     ,title CHAR(100)
7     ,channel_title CHAR(100)
8     ,category CHAR(100)
9     ,publish_time TIMESTAMP(2)
10    ,tags CHAR(1000)
11    ,views BIGINT
12    ,likes BIGINT
13    ,dislikes BIGINT
14    ,comment_count INTEGER
15    ,thumbnail_link CHAR(100)
16    ,comments_disabled BOOLEAN
17    ,ratings_disabled BOOLEAN
18    ,video_error_or_removed BOOLEAN
19    ,description CHAR(6000)
20 );
21
```

2.2. Carga de información

Para cargar la información a las relaciones de base de datos se utilizará la operación Bulk Insert para utilizar como origen los archivos CSV dispuestos con anterioridad, en este caso una para cada tabla.

El script de carga para las tablas de *CategoryRegions* y un ejemplo de *DataVideos_Canada* se presenta a continuación:

```

1 — Copia de datos de categorias por region
2 \COPY CategoryRegion(region , category_id , category) FROM '/home/eliel/
3 Posgrado/Semestre2/FundamentosBaseDeDatos/Tarea3/extraccionCSV_Category/
   RegionsCategorys.csv' WITH (FORMAT csv , DELIMITER ',', HEADER TRUE, NULL
   '');
4
5 — Copia de datos de videos Canada
6 \COPY DataVideos_Canada(video_id , trending_date , title , channel_title ,
   category_id , publish_time , tags , views , likes , dislikes , comment_count ,
   thumbnail_link , comments_disabled , ratings_disabled , video_error_or_removed
   , description) FROM '/home/eliel/Posgrado/Semestre2/
   FundamentosBaseDeDatos/Tarea3/limpiezaCSVs_Origen/CLEAN_CAvideos.csv'
   WITH (FORMAT csv , DELIMITER ',', HEADER TRUE, NULL '');
7 .
8 .
9 .
10

```

Los demás scripts de carga de información para tablas de videos se omiten ya que el código es muy similar que el de *DataVideos_Canada*.

El código para llenar la tabla *DataVideos_AllRegions* consiste en un comando *INSERT INTO TABLE*, ya que esta se va a poblar a partir de la información de todas las demás tablas usando *SELECT* y *UNION*:

```

1 — Insercion de datos a tabla de todas las regiones
2 INSERT INTO DataVideos_AllRegions
3
4 SELECT 'Canada' AS region , video_id , trending_date , title ,
5       channel_title , CR.category , publish_time , tags , views ,
6       likes , dislikes , comment_count , thumbnail_link ,
7       comments_disabled , ratings_disabled , video_error_or_removed ,
8       description
9 FROM DataVideos_Canada AS DV
10 INNER JOIN CategoryRegion AS CR
11 ON DV.category_id = CR.category_id
12 WHERE region = 'CA'
13 UNION
14
15 SELECT 'Germany' AS region , video_id , trending_date , title ,
16       channel_title , CR.category , publish_time , tags , views ,
17       likes , dislikes , comment_count , thumbnail_link ,
18       comments_disabled , ratings_disabled , video_error_or_removed ,

```

```
        description
19 FROM DataVideos_Germany AS DV
20 INNER JOIN CategoryRegion AS CR
21     ON DV.category_id = CR.category_id
22 WHERE region = 'DE'
23
24 UNION
25 .
26 .
27 .
28
```

3. Consultas

Se tienen 6 consultas que deben ser resueltas con la base de datos creada:

1. ¿Cuál es la región que tiene el video con mayor número de likes?
2. ¿Cuál es el video más antiguo?
3. ¿Cuál fue el video más popular por cada año?
4. ¿Cuál es la categoría con más videos?
5. ¿Cuál es el video más popular por categoría?
6. ¿Cuál es la categoría más popular por región?

A continuación se presenta el código de SQL utilizado para cada una de ellas con una breve explicación. Nota: Para todas las consultas solo se utilizó la tabla *DataVideos_AllRegions* que contiene toda la información de todas las regiones, y los resultados de las consultas se guardaron en archivos CSV para presentarlos de manera más legible en este reporte.

Los comandos usados para la ejecución de estas consultas se explicarán en la subsección *Medición de tiempos de ejecución de consultas*.

3.1. Resultados de consultas

1. ¿Cuál es la región que tiene el video con mayor número de likes?

En una subconsulta llamada *A*, se selecciona el mayor número de likes con *MAX(likes)* *AS maxlikes* que existen en la relación, y se aplica un *INNER JOIN* de *A* con *DataVideos_AllRegions* (alias *DV*) con los campos de *A.maxlikes = DV.likes* para obtener la región con el video con el mayor número de likes. La razón de este *JOIN* es porque puede existir más de una región con un video con la máxima cantidad de likes.

```
1 SELECT DV.region , DV.likes
2 FROM(
3     SELECT MAX(likes) AS maxlikes
4     FROM DataVideos_AllRegions
5 ) AS A
6 INNER JOIN DataVideos_AllRegions AS DV
7     ON A.maxlikes = DV.likes ;
8
```

Resultado:

1	region	likes
2	GreatBritain	5613827
3	USA	5613827

Figura 3.1: Resultado consulta 1.

2. ¿Cuál es el video más antiguo?

En una subconsulta llamada *A*, se selecciona la menor fecha $MIN(publish_time)$ AS *minpublishtime* que existe en la relación, se aplica un *INNER JOIN* de *A* con *DataVideos_AllRegions* (alias *DV*) con los campos de $A.minpublishtime = DV.publish_time$ para obtener el título del video con la menor fecha de publicación. La razón de este *JOIN* es porque puede existir más de un video con la menor fecha de publicación.

```
1 SELECT DV.title , DV.publish_time
2 FROM(
3     SELECT MIN(publish_time) AS minpublishtime
4     FROM DataVideos_AllRegions
5 ) AS A
6 INNER JOIN DataVideos_AllRegions AS DV
7     ON A.minpublishtime = DV.publish_time ;
8
```

Resultado:

1	title	publish_time
2	Budweiser - Original Whazzup? Ad	2006-07-23 08:24:11

Figura 3.2: Resultado consulta 2.

3. ¿Cuál fue el video más popular por cada año?

En una subconsulta llamada *A*, se selecciona el mayor número de vistas $MAX(\text{views})$ *AS maxviews* agrupado por año de publicación, esto se obtiene con $DATE_PART('YEAR', \text{publish_time})$ *AS anio*; se aplica un *INNER JOIN* de *A* con *DataVideos_AllRegions* (alias *DV*) con los campos de $A.\text{maxviews} = DV.\text{views}$ y $DATE_PART('YEAR', DV.\text{publish_time}) = A.\text{anio}$ para obtener los títulos de los videos con el mayor número de vistas por año.

```

1 SELECT A.anio , DV.title , A.maxviews AS views
2 FROM(
3     SELECT DATE_PART( 'YEAR' , publish_time) AS anio ,
4           MAX( views ) AS maxviews
5     FROM DataVideos_AllRegions
6     GROUP BY Anio
7 ) AS A
8 INNER JOIN DataVideos_AllRegions AS DV
9     ON DATE_PART( 'YEAR' , DV.publish_time) = A.anio AND
10        DV.views = A.maxviews
11 ORDER BY A.anio DESC;
12

```

Resultado:

1	anio	title	views
2	2018	Nicky Jam x J. Balvin - X (EQUIS) Video Oficial Prod. Afro Bros & Jeon	424538912
3	2017	YouTube Rewind: The Shape of 2017 #YouTubeRewind	169884583
4	2016	お昼寝の前にお話ししないと眠れない猫♥♥猫との会話を楽しむ動画 Conversation with a cat	1390596
5	2015	AC-130 Gunship Simulator - Convoy engagement	1176523
6	2014	Frida 'Dancing Around An Issue' (HD) - Salma Hayek, Ashley Judd MIRAMAX	787752
7	2013	Zombie - Cranberries MTV Unplugged	1250500
8	2012	Arvo Pärt - Tabula Rasa	731007
9	2011	Volar - Jaime Ciero	1674208
10	2010	the hell hole	727208
11	2009	Grey's Anatomy - (It's the End of the World) - In the sun - 2x16	264577
12	2008	Dolores O'riordan - her house in Canada	520808
13	2007	Bill is Dead The Fall	98981
14	2006	Budweiser - Original Whazzup? ad	258506

Figura 3.3: Resultado consulta 3.

4. ¿Cuál es la categoría con más videos?

En una subconsulta llamada *A*, se selecciona la categoría con mayor cantidad de videos, usando *COUNT(title) AS videoQty*, ordenando de forma descendente y limitando a 1 el resultado para asegurarse de que es la categoría con más videos.

En otra subconsulta llamada *B*, se seleccionan todas las categorías con su cantidad de videos, después se realiza un *INNER JOIN* de *A* con *B* con los campos de *A.videoQty = B.videoQty* para obtener la categoría con más videos. La razón de este *JOIN* es porque puede existir más de una categoría con la mayor cantidad de videos.

```
1 SELECT B.category , B.videoQty
2 FROM(
3     SELECT category , COUNT(title) AS videoQty
4     FROM DataVideos_AllRegions
5     GROUP BY category
6     ORDER BY videoQty DESC
7     LIMIT 1
8 ) AS A
9 INNER JOIN (
10     SELECT category , COUNT(title) AS videoQty
11     FROM DataVideos_AllRegions
12     GROUP BY category
13 ) AS B
14 ON A.videoQty = B.videoQty ;
15
```

Resultado:

1	category	videoqty
2	Entertainment	104567

Figura 3.4: Resultado consulta 4.

5. ¿Cuál es el video más popular por categoría?

En una subconsulta llamada *A*, se selecciona la categoría con el video con mayor cantidad de likes, usando $MAX(views)$ *AS maxviews* y agrupando por categoría.

En otra subconsulta llamada *B*, se seleccionan todas las categorías con el título del video que tengan la mayor cantidad de likes, agrupando esto por categoría y título de video. Después se realiza un *INNER JOIN* de *A* con *B* con los campos de $A.maxviews = B.views$ para obtener el video más popular por categoría. La razón de este *JOIN* es porque puede existir más de un video por categoría con la mayor cantidad de visitas.

```

1 SELECT B.category , B.title , B.views
2 FROM(
3     SELECT category , MAX(views) as maxviews
4     FROM DataVideos_AllRegions
5     GROUP BY category
6 ) AS A
7 INNER JOIN (
8     SELECT category , title , MAX(views) AS views
9     FROM DataVideos_AllRegions
10    GROUP BY category , title
11 ) AS B
12 ON A.maxviews = B.views
13 ORDER BY B.views DESC;
14

```

Resultado:

1	category	title	views
2	Music	Nicky Jam x J. Balvin - X (EQUIS) Video Oficial Prod. Afro Bros & Jeon	424538912
3	Entertainment	YouTube Rewind: The Shape of 2017 #YouTubeRewind	169884583
4	People & Blogs	To Our Daughter	62338362
5	Film & Animation	Selena Gomez - Back To You (Lyric Video)	54863912
6	Howto & Style	42 HOLY GRAIL HACKS THAT WILL SAVE YOU A FORTUNE	54155921
7	Comedy	Anitta & J Balvin - Downtown (Official Lyric Video) ft. Lele Pons & Juanpa Zurita	43460605
8	Science & Technology	Do You Hear Yanny or Laurel? (SOLVED with SCIENCE)	42799458
9	Sports	Real Life Trick Shots 2 Dude Perfect	29090799
10	Autos & Vehicles	Official Ram Trucks Super Bowl Commercial Icelandic Vikings We Will Rock You	25244097
11	Nonprofits & Activism	Suicide: Be Here Tomorrow.	24286474
12	Travel & Events	Turkish Airlines - 5 Senses with Dr. Oz	23932421
13	News & Politics	Aquí mi respuesta a los nuevos ataques del gobierno en mi contra.	21716633
14	Gaming	Yodeling Walmart Kid EDM Remix (OFFICIAL AUDIO) + DOWNLOAD LINK	18158133
15	Education	पुराने फ़िल्मों ग़द स़ीन, ज़िनके अग़ नय़ी फ़िल्मे भ़ी है फ़ोको bollywood double meaning scenes	12100921
16	Movies	Golak Bugni Bank Te Batua Full Movie (HD) Harish Verma Simi Chahal Superhit Punjabi Movies	7398655
17	Pets & Animals	野生動物の驚くべき瞬間2018! ライオンモンキーヒョウカンガルー	7220717
18	Shows	Пусть говорят - Диана Шурыгина шокирована освобождением Сергея Семенова. Выпуск от 15.01.2018	5167531
19	Trailers	TRAILER VIDEO	19845

Figura 3.5: Resultado consulta 5.

6. ¿Cuál es la categoría más popular por región?

En una subconsulta llamada *A1*, se selecciona la suma de visitas con *SUM(views) AS sumviews* agrupado por categoría y región; *A1* se utiliza en el *FROM* de otra subconsulta *A2*, la cual se usa para obtener la región con la cantidad de vistas máxima que exista en alguna de sus categorías (*MAX(sumviews) AS maxviews*). En otra subconsulta llamada *B2*, se selecciona la suma de visitas con *SUM(views) AS sumviews* agrupado por categoría y región. Después se realiza un *INNER JOIN* de *A2* con *B2* con los campos de *A2.maxviews = B2.sumviews* para obtener la categoría más popular de cada región.

```
1 SELECT B2.region , B2.category , B2.sumviews
2 FROM(
3     SELECT region , MAX(sumviews) AS maxviews
4     FROM(
5         SELECT region , category , SUM(views) AS sumviews
6         FROM DataVideos_AllRegions
7         GROUP BY region , category
8     ) AS A1
9     GROUP BY region
10 ) AS A2
11 INNER JOIN (
12     SELECT region , category , SUM(views) AS sumviews
13     FROM DataVideos_AllRegions
14     GROUP BY region , category
15 ) AS B2
16 ON A2.maxviews = B2.sumviews;
17
```

Resultado:

1	region	category	sumviews
2	Canada	Entertainment	13671215509
3	France	Music	5026447522
4	Germany	Entertainment	8102638694
5	GreatBritain	Music	170421287915
6	India	Entertainment	13326399551
7	Japan	Entertainment	1239934955
8	Korea	Entertainment	4350248043
9	Mexico	Music	4075106067
10	Russia	Entertainment	2085173930
11	USA	Music	40126286541

Figura 3.6: Resultado consulta 6.

3.2. Medición de tiempos de ejecución de consultas

Se medirá el tiempo de ejecución de las 6 consultas anteriormente expuestas, 30 veces en frío y 30 veces en caliente para cada una de ellas; después se calculará el promedio de tiempo de ejecución para cada consulta en frío, y en caliente.

Para automatizar esta tarea, se codificaron scripts en bash para ejecución de las consultas y guardar los tiempos de ejecución en archivos de texto plano, además, se escribió también un código en Python para calcular los promedios de las consultas a partir de los archivos de texto plano que almacenan los tiempos de ejecución de cada consulta.

Cada consulta se guardó en un script individual llamados Consulta1.sql, Consulta2.sql, etc.

- Tiempos de ejecución en caliente:

El proceso para ejecutar 30 veces cada consulta en caliente funciona a como sigue: primero se establece el formato de tiempo a segundos e impresión del tiempo real (línea 3), en la línea 6 se hace una ejecución de la consulta 1 para que exista información en el buffer de memoria. En la línea 11 se ejecuta la consulta 1 y se crea el archivo *tiemposHot.txt* el cual almacenará todos los tiempos de las ejecuciones de las consultas; el tiempo es medido escribiendo el comando *time* justo antes de la instrucción de ejecución de la consulta (todo esto agrupado con paréntesis), y después del cierre de paréntesis se escribe *1 > /dev/null* para silenciar la salida de la consulta (no mostrar resultado en la terminal) y *2 > tiemposHot.txt* para dirigir la salida del comando *time* al archivo de texto.

Después en un ciclo *for* se hacen las 29 ejecuciones restantes de la consulta 1 (líneas 15-19), esta vez escribiendo *2 > > tiemposHot.txt* al final de cada ejecución, para que el resultado del comando *time* se agregue al archivo de texto de resultados sin sobreescribirlo. Por último, con 2 ciclos *for* anidados se ejecutan las consultas restantes 30 veces cada una, el primer ciclo *for* es para iterar entre archivos de consultas (línea 24) y el segundo para cada ejecución (línea 26). Es importante mencionar, que después de terminar con las 30 ejecuciones de cada consulta, se agrega un guión (-) al archivo, para así separar los tiempos de cada consulta.

El código se presenta a continuación:

```

1 #!/bin/bash
2
3 TIMEFORMAT = %R
4
5 # Ejecucion para mantener datos en buffer
6 psql -U eliel -d youtubestatistics -f /home/eliel/Posgrado/Semestre2/
  FundamentosBaseDeDatos/Tarea3/scripts/SQL/Consulta1.sql 1>/dev/null
7
8 echo "Ejecucion para buffer"
9
10 # Ejecucion 1 de consulta 1 para crear el archivo de resultados tiemposHot.
   txt
11 (time psql -U eliel -d youtubestatistics -f /home/eliel/Posgrado/Semestre2/
  FundamentosBaseDeDatos/Tarea3/scripts/SQL/Consulta1.sql) 1>/dev/null 2>
  tiemposHot.txt
12 echo "Consulta 1, ejecuciones: 1"
13
14 # Resto de ejecuciones de consulta 1
15 for ((j = 2; j <= 30; j++))
16 do
17     (time psql -U eliel -d youtubestatistics -f /home/eliel/Posgrado/
  Semestre2/FundamentosBaseDeDatos/Tarea3/scripts/SQL/Consulta1.sql) 1>/
  dev/null 2>> tiemposHot.txt
18     echo "Consulta 1, ejecuciones: $j"
19 done
20
21 echo "-" >> tiemposHot.txt
22
23 # Resto de ejecuciones de todas las consultas
24 for ((i = 2; i <= 6; i++))
25 do
26     for ((j = 1; j <= 30; j++))
27     do
28         (time psql -U eliel -d youtubestatistics -f /home/eliel/Posgrado/
  Semestre2/FundamentosBaseDeDatos/Tarea3/scripts/SQL/Consulta$i.sql) 1>/
  dev/null 2>> tiemposHot.txt
29         echo "Consulta $i, ejecuciones: $j"
30     done
31
32     echo "-" >> tiemposHot.txt
33 done
34

```

- Tiempos de ejecución en frío:

Para el proceso de tiempos de ejecución en frío es prácticamente el mismo que el proceso para las ejecuciones en caliente, con la diferencia que al inicio no se hace una ejecución para almacenar datos en el buffer, y entre cada consulta se realiza el proceso de limpieza de buffer y caché del sistema, el cual consiste en: Desactivar el servicio de PostgreSQL, limpiar caché, e iniciar de nuevo el servicio de PostgreSQL; además, los resultados de tiempo son insertados al archivo *tiemposCold.txt*.

Se creó una función para realizar la limpieza de buffer y caché, llamada *limpiarBufferCache()* (línea 5), la cual es llamada al inicio del proceso, y entre cada una de las ejecuciones de las consultas. Dentro de la función, detiene el servicio de PSQL (línea 8), se ejecuta el script de limpieza de caché (línea 12), y se inicia el servicio de PSQL en la línea 16. Para cada una de estas instrucciones se les pasa la contraseña de usuario del sistema con *echo* y un pipeline, de manera que el sistema no tenga que esperar a que el usuario teclee la contraseña y todo sea automático.

El código se presenta a continuación:

```
1 #!/ bin / bash
2
3 TIMEFORMAT=%R
4
5 limpiarBufferCache ()
6 {
7     # Apagar servicio psql
8     echo "eliel123" | sudo -S service postgresql stop;
9     echo "Servicio psql detenido"
10
11     # Limpiar cache y buffer
12     echo "eliel123" | sudo -S ./clear_cache
13     echo "Buffer y cache liberados"
14
15     # Iniciar servicio psql
16     echo "eliel123" | sudo -S service postgresql start;
17     echo -e "Servicio psql iniciado\n"
18 }
19
20 # LIMPIAR BUFFER Y CACHE
21 limpiarBufferCache
22
23 # Ejecucion 1 de consulta 1 para crear el archivo de resultados tiemposCold
24 .txt
25 (time psql -U eliell -d youtubestatistics -f /home/eliell/Posgrado/Semestre2/
```

```

FundamentosBaseDeDatos/Tarea3/scripts/SQL/Consulta1.sql) 1>/dev/null 2>
tiemposCold.txt
25 echo "Consulta 1, ejecuciones: 1"
26
27 # LIMPIAR BUFFER Y CACHE
28 limpiarBufferCache
29
30 # Resto de ejecuciones de consulta 1
31 for ((j = 2; j <= 30; j++))
32 do
33     (time psql -U eliel -d youtubestatistics -f /home/eliel/Posgrado/
Semestre2/FundamentosBaseDeDatos/Tarea3/scripts/SQL/Consulta1.sql) 1>/
dev/null 2>> tiemposCold.txt
34     echo "Consulta 1, ejecuciones: $j"
35
36     # LIMPIAR BUFFER Y CACHE
37     limpiarBufferCache
38 done
39
40 echo "-" >> tiemposCold.txt
41
42 # Resto de ejecuciones de todas las consultas
43 for ((i = 2; i <= 6; i++))
44 do
45     for ((j = 1; j <= 30; j++))
46     do
47         (time psql -U eliel -d youtubestatistics -f /home/eliel/Posgrado/
Semestre2/FundamentosBaseDeDatos/Tarea3/scripts/SQL/Consulta$i.sql) 1>/
dev/null 2>> tiemposCold.txt
48         echo "Consulta $i, ejecuciones: $j"
49
50         # LIMPIAR BUFFER Y CACHE
51         limpiarBufferCache
52     done
53
54     echo "-" >> tiemposCold.txt
55 done
56

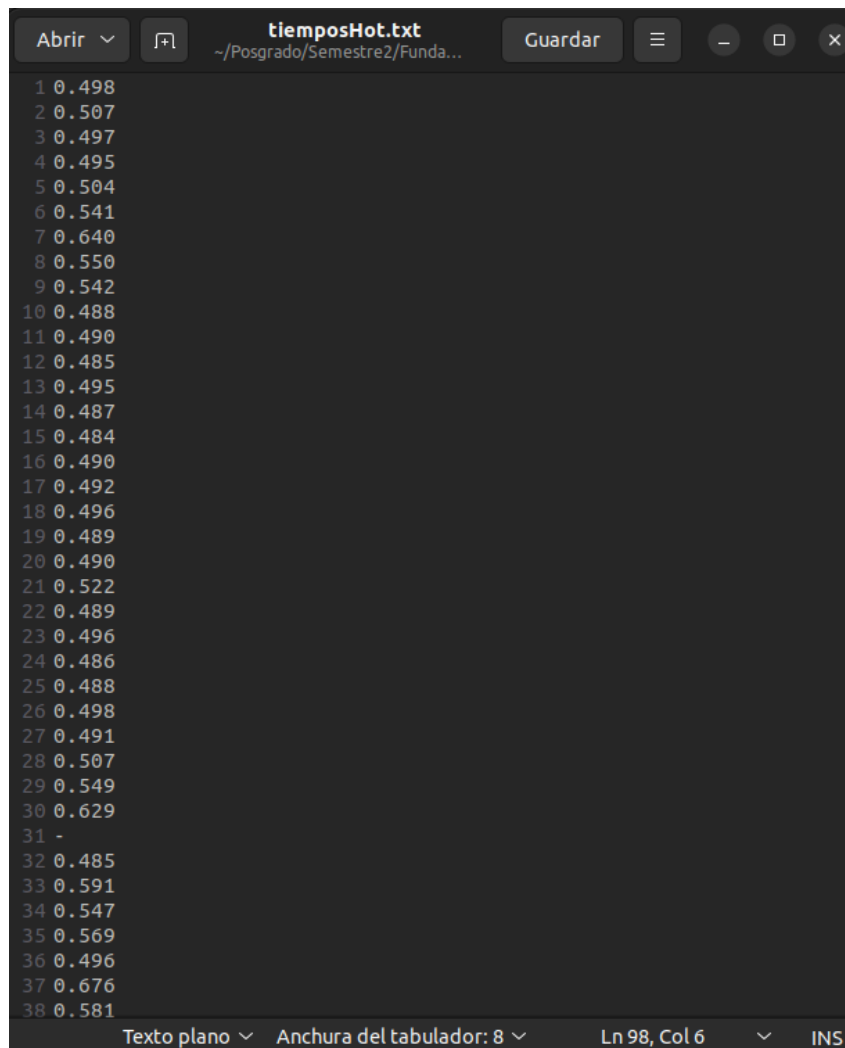
```


Dentro de la función de limpieza se ejecuta el script *clear_cache*, el cual primero revisa si se está ejecutando con permiso de root en la línea 3, después elimina los datos que existen en la caché con las instrucciones restantes.

El código se presenta a continuación:

```
1 #!/bin/bash
2
3 if [[ $(id -u) -ne 0 ]] ; then echo "Please run as root" ; exit 1 ; fi
4 sync; echo 1 > /proc/sys/vm/drop_caches
5 sync; echo 2 > /proc/sys/vm/drop_caches
6 sync; echo 3 > /proc/sys/vm/drop_caches
7
```

Los resultados de los códigos que almacenan los tiempos de ejecución de las consultas, finalmente quedan en esta disposición:



The screenshot shows a text editor window titled 'tiemposHot.txt' with a file path of '~/.Posgrado/Semestre2/Funda...'. The editor contains a list of 38 lines, each representing an execution time. The times are mostly between 0.48 and 0.6 seconds. Line 31 contains a hyphen '-'.

Line	Time
1	0.498
2	0.507
3	0.497
4	0.495
5	0.504
6	0.541
7	0.640
8	0.550
9	0.542
10	0.488
11	0.490
12	0.485
13	0.495
14	0.487
15	0.484
16	0.490
17	0.492
18	0.496
19	0.489
20	0.490
21	0.522
22	0.489
23	0.496
24	0.486
25	0.488
26	0.498
27	0.491
28	0.507
29	0.549
30	0.629
31	-
32	0.485
33	0.591
34	0.547
35	0.569
36	0.496
37	0.676
38	0.581

Figura 3.7: Ejemplo de tiempos de ejecución almacenados.

- Cálculo de promedios de tiempos de ejecución:

Para calcular los promedios de ejecución a partir de los archivos de texto generados en el apartado anterior se usó un script en Python.

Primero se define la ruta de los archivos, una lista que guarda los nombres de los archivos y otra lista que guardará una sublista para tiempos en caliente, y otra sublista para tiempos en frío. Con ayuda de un ciclo *for*, abrimos ambos archivos y se insertan en *listaTiempos*.

```
1 import numpy as np
2
3 ruta = '/home/eliel/Posgrado/Semestre2/FundamentosBaseDeDatos/Tarea3/
    scripts/bash/'
4
5 nombresArchivos = ["tiemposHot.txt", "tiemposCold.txt"]
6
7 # Lista de listas, primer sublista tiemposHot, segunda sublista tiemposCold
8 listasTiempos = []
9
10 # Abrir archivos y guardar su contenido en listasTiempos
11 for nombreArchivo in nombresArchivos:
12     with open(ruta + nombreArchivo, 'r') as archivo:
13         listasTiempos.append(archivo.read().split("\n"))
14
```

Ya que se tienen los datos dentro de la lista, se crean otras dos listas, *listaTiemposPorConsulta* que almacene los tiempos por consulta y por tipo de consulta i.e. consulta 1 en frío, consulta 2 en frío, consulta 1 en caliente, etc. Y *listaPromedios* que almacenará todos los promedios en frío en una sublista, y todos los promedios en caliente en otra sublista.

Después, con dos ciclo *for* anidados, iteramos sobre cada uno de los elementos de *listasTiempos*, añadiéndolos a las sublistas correspondientes (líneas 10-26), cuando el caracter de guión (-) es encontrado, quiere decir que se terminaron de analizar los tiempos de alguna consulta en específico, por lo que hay que continuar añadiendo los tiempos de la siguiente consulta en otra sublista (líneas 12-22). Además, cuando se termina de añadir los tiempos de cada consulta, se calcula el promedio de esta en la línea 14, así se aprovechn los ciclos *for* para hacer este cálculo también y agregarlos a *listaPromedios*.

Por último, se imprimen los resultados en las líneas 29-35 y se calcula la relación de diferencia de velocidad de las consultas en caliente con respecto a las consultas en frío (líneas 38-39).

El código se presenta a continuación:

```
1 # Primera sublista para tiemposHot y segunda sublista para tiemposCold
2 listaTiemposPorConsulta = [[[]], [[]]]
3 listaPromedios = [[], []]
4
5 # j para hot y cold, i para elementos dentro de las listas (tiempos de
   # ejecucion)
6 for j in range(2):
7     i = 0
8
9     # Iterar por todos los elementos
10    for elemento in listasTiempos[j]:
11        # Si elemento es "-", entonces ya se leyeron los 30 resultados de
   # una consulta
12        if(elemento == "-"):
13            # Agregar el promedio de la consulta i a la lista de promedios
14            listaPromedios[j].append(round(np.average(
15                listaTiemposPorConsulta[j][i]), 3))
16
17            i += 1
18
19            # Fin del archivo
20            if(i == 6):
21                break
22
23            # Se agrega nueva lista a la lista j
24            listaTiemposPorConsulta[j].append([])
25            continue
26        else:
27            # Agregar elemento a la lista j, i.
28            listaTiemposPorConsulta[j][i].append(float(elemento))
29
30    # Imprimir resultados
31    if(j == 0):
32        print("Promedios en tiempos de ejecucion en caliente (segundos)")
33    else:
34        print("Promedios en tiempos de ejecucion en frio (segundos)")
35
36    for k in range(1,7):
37        print("Consulta " + str(k) + ": ", listaPromedios[j][k-1])
38        print("\n")
39    print("Relacion de velocidad, consulta en caliente con respecto a consulta
   # en frio")
40    for k in range(1,7):
41        print("Consulta " + str(k) + ": " + str(round(((listaPromedios[1][k-1]/
   # listaPromedios[0][k-1]) - 1) * 100, 2)) + "% mas rapido"))
```

Los resultados de promedios se presentan en la siguiente figura.

```
Promedios en tiempos de ejecución en caliente (segundos)
Consulta 1: 0.51
Consulta 2: 0.577
Consulta 3: 0.887
Consulta 4: 0.59
Consulta 5: 4.063
Consulta 6: 0.643

Promedios en tiempos de ejecución en frío (segundos)
Consulta 1: 2.596
Consulta 2: 2.656
Consulta 3: 2.2
Consulta 4: 2.579
Consulta 5: 5.8
Consulta 6: 2.73
```

Figura 3.8: Promedios de tiempos de ejecución.

Las consultas en caliente son más rápidas debido a que los bloques de disco que el sistema necesita para computarlas se guardan en frames de buffer de la memoria RAM, para hacer esto más ilustrativo, se presenta también la relación de diferencia de velocidad de las consultas en caliente con respecto a la consultas en frío.

```
Relación de velocidad, consulta en caliente con respecto a consulta en frío
Consulta 1: 409.02% más rápido
Consulta 2: 360.31% más rápido
Consulta 3: 148.03% más rápido
Consulta 4: 337.12% más rápido
Consulta 5: 42.75% más rápido
Consulta 6: 324.57% más rápido
```

Figura 3.9: Relación de diferencia de velocidad.

4. Arquitectura computacional

El equipo utilizado es una laptop Dell Latitude 5480, con Intel Core i5-6300U, 4 núcleos con cachés L1 de 64 kB, L2 de 512 kB, y L3 de 3 MB. La memoria principal es de 8GB de RAM DDR4 de 2400 MHz. Como dispositivo de almacenamiento se tiene un SSD SanDisk X400 M.2 2280 con 256 GB de capacidad, con una velocidad de lectura/escritura de hasta 545/520 MB/s.

Los frames de buffer que tiene esta instancia de PostgreSQL son 16,384 cada uno midiendo 8 kB, de los cuales cada consulta ejecutada inserta bloques de disco en ellos para poder tener una ejecución más rápida.