## Objective:

This project was to create a platform to observe correlations within the interaction between steams’ user base and all the video games they play.

## Troubles n’ Turmoil

### Initial Problem

File is a MySQL dump that is inaccessible due to scale (160gb file with data on 108 million users).

We used a lightweight program to access the database and divide the dump into smaller files for each tables insert. This however did not solve accessibility due to Memory Issues for all group members.

### Insufficient Memory

In order to read the MySQL files aa python script was made to do it in chunks. We created a sample file containing a set number of lines, trimmed off the last incomplete insert statement, then save file. Using that we now had a working chunksize to pass through the function.

Using the new chunksize, we could pull the schema out of the tables and delete it for later editing during sqlite conversion. Since the insert statements are same those could be left alone.

When the script runs, a new sqlite schema would be inserted which would allow all the sql files to be converted into workable sqlite files. Using chunksizes allowed us to do this for the 60GB files as well bypassing the memory issues.

### User Data Shenanigans

Since data contains user defined usernames, some people created names that would purposefully break code. This meant if a single person created a name ending in "\\" or "','" the insert statement would fail, and we would lose about 18,000 records. These errors dropped about 50% of the data.

To solve this a new script had to be created that will be able to find the errors and replace them with a correction\*.

\*Note: This process has its own issues but saved 95% of the data.

Running this script would output a new sqlite file for inserting into database.

### Big Data Issues

Further memory issues were encountered when running the large sqlite files to get the dumped data back into a database.

To bypass the memory issue we had to download sqlite3.exe and put file into working directory. From powershell we could connect to the sqlite.exe and run the files through the exe instead of terminal. This allowed us to read the large sql files into our new database using command line. This process allows a computer with low memory to bypass the memory requirements of big data.

### Huge Database

As expected, the result of combining all the sql files created an unworkable database due to memory errors.

As a result, whenever working with large tables we had to do so through Powershell. Smaller tables were still accessible using vscode. Significant amount of data unneeded by our team was dropped out of the database in order to create a workable 25GB file.

### Huge Database Pt.2

Database did not decrease after cleaning, pathing issues using multiple drives.

We learned that when data is dropped from an SQL database, the data is not fully deleted. The database instead stores values into deleted tables to preserve the allocated space. After cleaning database, we needed to run the vacuum command to free up the preserved disk space, finally resulting in a proper 25GB database.

This process requires extra space about equal to database size in order to run. When doing this from two different locations a new temporary folder needed to be created. We had to use “$env.TMP ‘directory’” with powershell, then running vacuum from there.

### Big Tables, Slow Queries

Games\_1 and Friends use very large tables with slow query times.

From this point we needed to create indicies for all the tables on AppID and SteamID for faster queries. To improve speeds further, new tables were joined, and the use of views was implemented for faster callbacks.

### Choropleth GeoJSONA and Steam Locations:

There is not geojson for the steam data to plot on to a map.

We had to find a geojson for all the countries in the world. https://github.com/johan/world.geo.json/blob/master/countries.geo.json

Then get a csv file that contains ISO Alpha 2 and ISO Alpha 3 data. https://gist.github.com/tadast/8827699#file-countries\_codes\_and\_coordinates-csv

From there make a new table for Steam's LocCountryCode's ISO Alpha 2 on the csv file’s Alpha 2, in order to get table with Steam locations in Alpha 3 format, for pairing with the GeoJSON’s Alpha 3.

### Flask & Dash Woes

When finding out that Flask doesn’t have callbacks, we had to look into Dash. However Dash doesn’t have “.route” so we hooked dash to flask in order bypass both of these issues.

## Assets:

160GB database with video game and player data on Steam.

<https://academictorrents.com/details/eba3b48fcdaa9e69a927051f1678251a86a546f3>

GeoJSON template for every country.

<https://github.com/johan/world.geo.json/blob/master/countries.geo.json>

Alpha 2 and Alpha 3 codes for every country:

<https://gist.github.com/tadast/8827699#file-countries_codes_and_coordinates-csv>

## Tools, Libraries, Databases, Etc:

Flask, Dash, Dash Components, Dash Bootstrap, Plotly, Python, HTML, CSS, Javascript, Bootstrap, Sqlite