CSC343 Project Phase 2:

What does it take to get to the top? An analysis on speedrun.com data 🏃‍♀️

Fabian, Neeco[1] Mathew, Steven[1]

[1] *University of Toronto*

Speedruns are playthroughs of games with the purpose of completing it as quickly as possible. In Phase 2, we explored data from https://speedrun.com about games, players, and runs. Accordingly, we refined our schema and used `speedrun.com`'s REST API to obtain data. We prepared it for import into SQL, where the data was inserted into the tables we will use in Phase 3 queries.

Design Decisions

One of the TA's concerns in Phase 1 were the many possible questions that we will later answer with our queries. We narrowed our inquiry to three primary questions and posed smaller possible sub-questions that we may use to answer the primary questions.

1. Who are the best of the best?
    - Which speedrunners are most talented?
    - Which countries excel in the leaderboards?
    - Who are rivals? Which players tend to place in the same leaderboards together?
2. Are speedrunners improving over time?
    - Are average run durations getting shorter for a game?
    - How often do the top players of a leaderboard change?
    - How many attempts on average does it take to get a world record?
3. Do some players have [unfair] advantages?
    - Do games played on emulators offer speedrunners an edge?
    - Can certain region-specific versions of games yield faster times?
    - Are some examiners biased? Do some examiners examine certain successful players often?

We will continue to refine these questions as we begin Phase 3. As we expressed our schema in DDL, we made the following important design decisions:

- The TA indicated that some of the Game-related tables in Phase 1 could be combined: `Game`, `GameRegion`, `GameCategory`, `GamePlatform`, `GameSeries`. To simplify our schema, we removed tables `GameCategory`, `GamePlatform`, `Platform`, `GameSeries`, and `Series` since we will not use their data. Further, we replaced the `GameRegion` table with the `regionName` attribute in `Game`. We recognize the functional dependency that the game id `GID` and `gameName` have, which would normally necessitate their own

table. However, including `regionName` was appropriate because `GID` and `regionName` are commonly used together, so this will reduce the frequent joining that would have been needed with the old schema. *Note:* we created `RegionDomain` - which stores valid `regionNames` - instead of a table after noticing that the data had a small, constant set of possible `regionNames`.

- Since `user` is a reserved word, the `User` table was renamed to `Player`. Though minor, we also renamed `RunCategory` to `RunType`.
- While viewing `speedrun.com` data, we noticed that some speedrunners do not have accounts, and consequently, are missing information such as a unique `PID` and their country's `CID`. We decided to exclude these players and their runs to avoid `NULL` values for the attributes in `Player`.
- Not all speedruns were verified so we removed unverified entries, which eliminates the need for the `isVerified` boolean in `Run` and possible `NULL`s for the examiner's ID. This contributes to the validity of our data as we are left with just the verified runs. Also, we removed runs completed by more than one player, so we no longer need the `UserRun` table. Now, we also store the `PID` of the player who completed the speedrun directly inside `Run`. Lastly, we did not include runs whose `regionName` was invalid. Otherwise, there is no way to determine which version of the game the player used. These changes helped us reach our goal of simplifying the schema to better capture our focused queries.
- Due to the deletions resulting from the design choices and data cleanup, the placement of a speedrun that we retrieved from `speedrun.com` may no longer be accurate when compared to the remaining runs in SQL. Therefore, in Phase 3, when we have queries that require placement in a leaderboard, we will use SQL to assign an accurate rank to the runs - grouped by game, region-specific version, and run type.

Cleaning Process / Crawling

For games:

- Since we needed to scrape the data ourselves from the REST API, we cleaned the data as soon as we fetched it. Please note that the `.py` file is very messy!
- To begin, we gathered as many games (with regions) as we could and found 28,986 "games". There aren't exactly 28,986 games, but 28,986 game/region pairings. For example, the game "Zork: Grand Inquisitor" is in two regions USA / NTSC and EUR / PAL. These versions would count as two rows in the table. The process is as follows:
- The API has handy tooling like pagination and offsets, so we used this to iterate through as many pages as possible, reading a maximum of 200 per page, and exiting when none could be read (in which case, the string "data" would not appear in the JSON). The endpoint for this is

  `https://www.speedrun.com/api/v1/games?max={PAGE}&offset={offset}`,

  and we shift `offset` by (`pages * PAGE`), where `pages` increases by 1 for each game and `PAGE = 200`.
- We iterate through each of the results per page, and ignore games without a region listed.
- Then using `pd.json_normalize`, we can transform a nested JSON into a table. We hold on to a list `frames` so that we can add these data frames to it.
- We filter only for columns we want: `GID`, `gameName`, `regionName`.
- As an aside, `regionName` does not come from the table, but `regionId` does. But we have a dictionary mapping `regionId` to `regionName`, defined by

```
REGIONID_TO_STRING = {
    "ypl25l47": "BRA / PAL",
    "mol4z19n": "CHN / PAL",
    "e6lxy1dz": "EUR / PAL",
    "o316x197": "JPN / NTSC",
    "p2g50lnk": "KOR / NTSC",
    "pr184lqn": "USA / NTSC",
}
```

- This was found by making a GET request to https://www.speedrun.com/api/v1/regions.
- Further, since the regions are in a list, we can use panda's `explode` on each data frame and drop rows with missing values. We add this data frame to `frames`.
- Finally, we concatenate all the frames together for each game (and region) and save this data frame to a CSV using `pd.DataFrame.to_csv`.

Now, if we were to crawl through all runs of every game in our table, this dataset would be much too large for `teach.cs` to handle and the sysadmins will complain. It turns out, most of the games don't have many runs, ideally, we want games with many runs. Accordingly, we will iterate through every game and if the number of runs in a game is < 100, we will say that the game is not popular enough and we should skip this game.

- For each game, we will make a request to this endpoint:

  https://www.speedrun.com/api/v1/games?game={game}max=150
  status=verifiedorderby=verify-datedirection=desc

  Here, we are getting 150 newly verified runs per `game`.
- At this point, we will check if the total runs of this game is < 150, and if it is, skip to the next game. We'll stop looking once we've found 250 games. This means our table will have *at most* 37,500 rows.
- Then for each JSON in the request, we will filter out games where there is no date, no examiner, no region, keeping only solo runs. Then we grab only attributes we need according to our schema: `RUNID, RUNTYPEID, GID, duration, submissionDate, isEmulated, EID, regionName, PID`. If after going through all the runs of a game, at least 50 runs remain after filtering the 150 runs, then it's added as a game in our list.
- Meanwhile, we add all the players and examiners we encountered to a set of players and add this data frame to a `frames` list, ensuring that this player has a country code. Also, we add all the run types we've seen so far to a list.
- Note that we are again using `json_normalize` for each run and concatenating the runs together at the very end using `pd.concat(frames)`.
- Finally, we save the players, runs, and run types data frames to CSVs. All in all, we're left with a diverse group of games, each having between 50 and 150 runs and 21,998 runs altogether. We're also left with 4030 unique players and 1203 run types.

For players:

- We go through the list of players we saved from the previous request and make a GET request to

  https://www.speedrun.com/api/v1/users/user

- We normalize the JSON into a data frame and grab only their ID, international name, and country code, renaming each attribute to match our schema.

- Finally, we export to a CSV. We searched for the regex pattern `([a-z]{2})/\w*` and replaced instances with `$1` to find country codes that contained a regional code appended to the end. For example, "ca - qc" represented Quebec, Canada. In these cases, we only kept the country code, like "ca".

For each run type:

- We do the exact same process as players except make a request to the endpoint

  https://www.speedrun.com/api/v1/categories/rt,

  for each `rt` in the run type IDs, grabbing only the ID and name of the run, and renaming ID and name to fit our schema.

More cleaning for runs and run types:

- Unfortunately, there were some runs in the `Run` table that strangely didn't appear in the run type endpoint! So we made a compromise and only took runs where the run type ID appeared in both.
- We also used the regex `([,].+)[,](.*)` to find `runTypeNames` that contain a comma, and removed the comma by replacing the pattern with `$1$2`.

For countries:

- We obtained valid ISO Alpha-2 country codes and names from another source: https://gist.github.com/tadast/8827699.
- After removing unused columns, we used regex to make country codes lower case and to delete rows with duplicate `CID`s due to variants of their `countryName`.

Note: in all these queries, we sleep to avoid rate limiting.