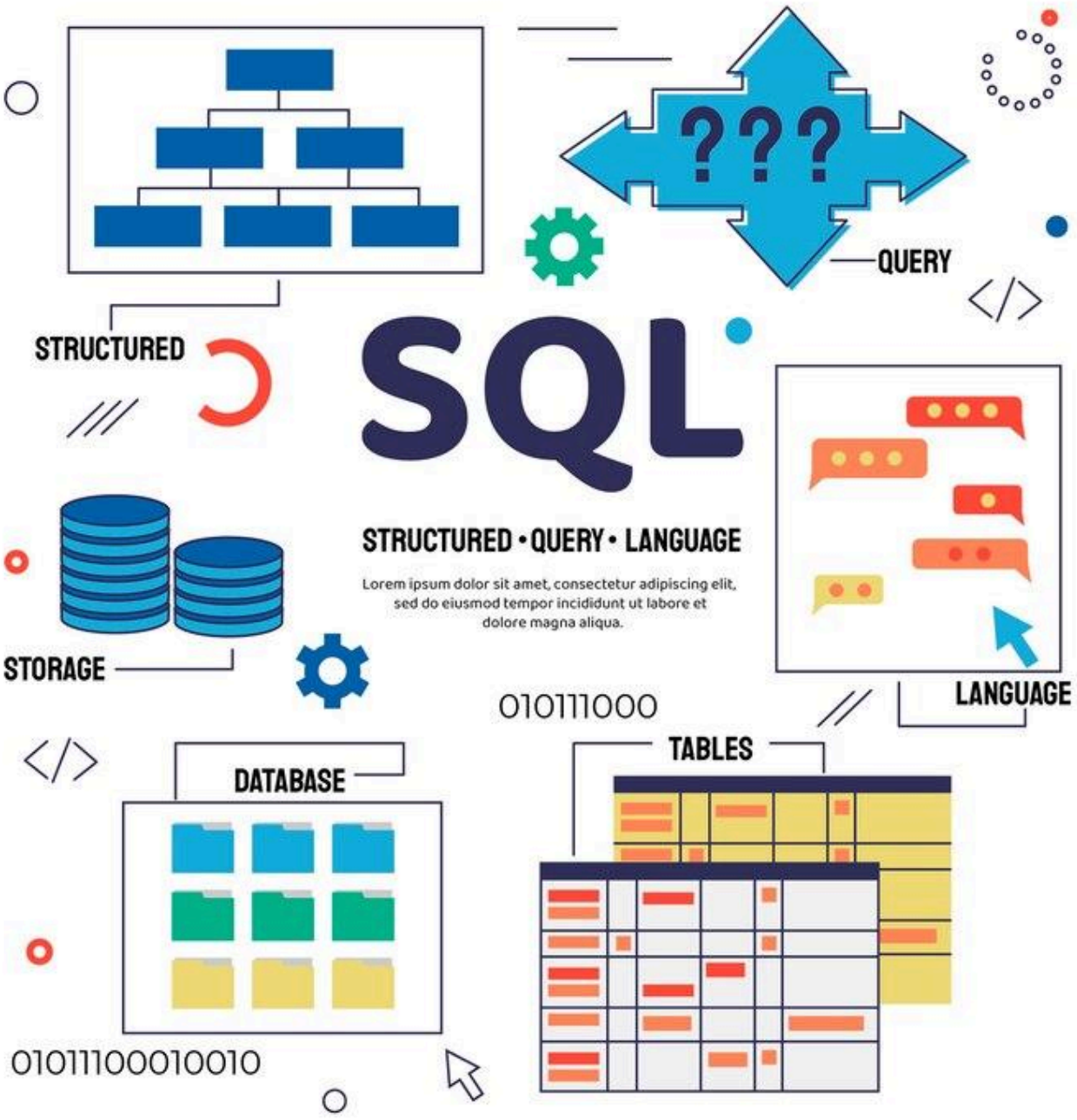
# Cleaning My Gaming Datasets

**Sean B. Coates**




# Introduction

This is meant as a proof-of-knowledge project. It showcases my skills in PostgreSQL, knowledge of databases, schemas, table creation, data exploration, and data cleaning.

I first collected the datasets I needed for both of my projects on table-top games and video games. I converted all collected datasets to CSV files in order to import them into pgAdmin 4. I created a new database in pgAdmin 4 titled, “*gaming*,” with two schemas titled, “*videogames*,” and “*boardgames*.”

# Video Games Cleaning

For my video games project, I collected data from three datasets. The first one I retrieved is video game industry revenue per year and broken down by platform types from The Visual Capitalist, which states that the original data came from Pelham Smithers. I created a CSV file with the data from the article.

Next I retrieved two datasets from Kaggle. The first one is Top Video Games 1995-2021 Metacritic which contains Metacritic website data on video games, such as title, platform, and date released, as well as Metacritic and users' scores. The third dataset is an updated version of the previous dataset from Metacritic Games, which should contain the same data plus more. It contains data on video games from 1986 to 2023. Both datasets were provided in CSV files.

In the schema titled, "*videogames*," I created 3 tables corresponding to the 3 datasets. I then used PostgreSQL in pgAdmin to explore the data, clean the data, and save the final tables I needed for my projects on analyzing of video games.

The following are the tables with their field names, data types, constraints, and short descriptions of what they should contain.

Created *industry_revenue* table (52 records):

| Fields: | Types & Constraints: |
|---|---|
| year | int, primary key, from 1971 to 2022 |
| arcades | int, arcades revenue for specified year rounded to nearest billion USD |
| consoles | int, consoles revenue for specified year rounded to nearest billion USD |
| handhelds | int, handhelds revenue for specified year rounded to nearest billion USD |
| pc | int, PC revenue for specified year rounded to nearest billion USD |
| mobile | int, mobile revenue for specified year rounded to nearest billion USD |
| vr_ar | int, VR and AR revenue for specified year rounded to nearest billion USD |
| total | int, total industry revenue for specified year rounded to nearest billion USD |

Not every year has a corresponding revenue within each field (null values). Fields such as vr_ar and mobile have many null values because their platforms didn't enter the industry until years later.

Created *games_old* (18,800 records) and *games_new* (142,417 records) tables (both tables contain identical fields):

| Fields: | Types & Constraints: |
|---|---|
| title | text, game title |
| platform | text, platform the title was released on and the score corresponds to |
| release_date | date, date of title release in yyyy-mm-dd format |
| summary | text, description of game from publisher |
| meta_score | numeric(3), critics score from metacritic website from 1 to 100 |
| user_score | numeric(3,1), average of users' score from metacritic website 0.1 to 10 |

The *industry_revenue* table does not need to be cleaned. The *games_old* and *games_new* tables need to be cleaned and then combined into a single table for my project.

Some game titles are repeated on the same table because they are listed under different platforms. This is normal, and those values need to be kept. I will refer to these title-platform combinations as TPCs from here on. It's possible they also have different release dates, and will almost certainly have different scores, all of which is necessary data for this project. As summary is the least important field, and not necessary for my projects, then it won't be included.

To start I should check if there are any TPCs from games_new and games_old which do not contain either a meta or user score. Both scores are needed for my projects, and the games which don't contain one or the other are likely to be very niche games which could skew our data (niche game fans tend to rate them very favorably, and those who aren't fans aren't likely to rate them at all). I use a WHERE function to filter out null values.

**PostgreSQL:**

```
SELECT title, platform, release_date, meta_score, user_score
FROM games_new
WHERE meta_score IS NOT NULL AND user_score IS NOT NULL;
```

Now I have an idea of the size of the dataset I'm working with, 24,248 records returned from *games_new* and 17,435 from *games_old*.

Next I want to find out what is duplicated in the *games_new* and *games_old* tables, and only keep the TPCs from games_new, but still include the games which appear solely on the games_old table. I use a UNION with both tables. This removes the duplicates in games_old, and keeps the original record from the games_new table.

**PostgreSQL:**

```
SELECT title, platform, release_date, meta_score, user_score
FROM games_new
WHERE title IS NOT NULL AND platform IS NOT NULL AND release_date IS NOT NULL
    AND meta_score IS NOT NULL AND user_score IS NOT NULL
UNION
SELECT title, platform, release_date, meta_score, user_score
FROM games_old
WHERE title IS NOT NULL AND platform IS NOT NULL AND release_date IS NOT NULL
    AND meta_score IS NOT NULL AND user_score IS NOT NULL
    ORDER BY title;
```

This gives me 33,583 records, but it's not the final dataset to use. Some of the titles seem to be released for the same platform twice. This isn't a re-release, nor a mistake in the data. Some developers release a new game in a series with an identical name to a previous edition (Need For Speed: Most Wanted 2005 and 2012 editions, both released on the Xbox 360 and are completely different games). In cases like this I am working with a TPC and release date combination. Fortunately for me, the UNION function is keeping all of them.

There is another issue though. Some of the TPC and date combinations (TPC-Ds) do appear twice, but with different scores in either meta or user fields. This is likely from the TPC-Ds being included in the *games_new* dataset, but with an updated meta score or user score after the update. As previously stated, the user scores are an average, so it's very likely to change for newer or more popular releases as time goes on.

First I want to check if there are duplicates in either table, I will run the following query for both tables to check this.

**PostgreSQL:**

```
SELECT title, platform, release_date, COUNT(*)
FROM games_old
WHERE meta_score IS NOT NULL AND user_score IS NOT NULL
GROUP BY title, platform, release_date
HAVING COUNT(*) > 1
```

When running this query on *games_new*, nothing is returned. However, *games_old* returns 58 records. For my final UNION, those need to be filtered out to just one copy, and then also compared to *games_new* to keep only the *games_new* version. I will need to use a DISTINCT function and a subquery with concatenation on my UNION function to ensure no TPC-Ds are duplicated. Where they are duplicated, the *games_new* copy is kept, excluding the ones from *games_old*.

**PostgreSQL:**

```
CREATE TEMP TABLE temp_comb AS
SELECT title, platform, release_date, meta_score, user_score
FROM games_new
WHERE meta_score IS NOT NULL AND user_score IS NOT NULL
UNION
SELECT DISTINCT ON (title, platform, release_date) title, platform, release_date, meta_score, user_score
FROM games_old
WHERE meta_score IS NOT NULL AND user_score IS NOT NULL
    AND title || platform || release_date NOT IN (SELECT title || platform || release_date  FROM
games_new);
```

Now I need to check for duplicate TPC-Ds to make sure I ran the proper query.

**PostgreSQL:**

```
SELECT title, platform, release_date, COUNT(*)
FROM temp_comb
GROUP BY title, platform, release_date
HAVING COUNT(*) > 1;
```

No records are returned. Now that all of my data cleaning is finished a new table will be created with the data from the temporary table. I need a primary key, so I will first create a new table called *games_clean*, and create a new field called game_id with a serial data type to auto-generate a primary key.

**PostgreSQL:**

```
CREATE TABLE games_clean
(game_id SERIAL PRIMARY KEY,
title TEXT,
platform TEXT,
release_date DATE,
meta_score numeric(3),
user_score numeric(3,1));
```

Now I will insert into my new table, *games_clean,* all of the data from my temporary table, *temp_comb*. I want to put the oldest games at the top, so I will add an ORDER function.

**PostgreSQL:**

```
INSERT INTO games_clean (title, platform, release_date, meta_score, user_score)
SELECT title, platform, release_date, meta_score, user_score
FROM temp_comb
ORDER BY release_date, title, platform, meta_score, user_score;
```

Our new table contains 25,897 records with the following fields:

| Fields: | Types & Constraints: |
|---|---|
| game_id | serial, primary key |
| title | text, game title, can be duplicated |
| platform | text, platform the title was released on |
| release_date | date, date of title release in yyyy-mm-dd format |
| meta_score | numeric(3), critics score from metacritic website from 1 to 100 |
| user_score | numeric(3,1), average of users' score from metacritic website from 0.1 to 10 |

None of the values in the new table will be null, as every field is necessary for my projects. For my projects I will be using the *games_clean* and *industry_revenue* tables to make aggregations and get insights into the video game industry. My *games_clean* table contains titles from September 30, 1994 to March 6, 2023.

# Table-Top Games Cleaning

I'm using data from the website Board Game Geek, the largest online community of board game enthusiasts. It has a lot of statistical data from users, as well as detailed descriptions of the games. The statistical data, e.g., ranking, number of user ratings, etc., are always changing. That data I downloaded on January 15th, 2024, so it's up-to-date for my project.

For my table-top games project, I collected and cleaned 3 datasets. I downloaded the BoardGameGeek items dataset and Board Games Dataset from Kaggle. I also downloaded the BoardGameGeek's CSV files from the BoardGamesGeek website, which contains statistical data on every title. It is updated upon request for the files.

In my database I created the following 4 tables under the "*boardgames*" schema.

Created *bgg_ranks* table (151,199 records):

| Fields: | Types & Constraints: |
|---|---|
| game_id | bigint primary key |
| name | text not null |

| | |
|---|---|
| year_published | int |
| rank | int |
| rating | numeric(8,6) |
| users_rated | int |
| abstracts_rank | int |
| cgs_rank | int |
| childgames_rank | int |
| familygames_rank | int |
| partygames_rank | int |
| stratgames_rank | int |
| thematic_rank | int |
| wargames_rank | int |

Created *bgg_details* table (90,400 records):

| Fields: | Types & Constraints: |
|---|---|
| game_id | bigint primary key |
| name | text not null |
| game_type | text not null |
| year_published | int |
| min_players | int |
| max_players | int |
| min_playtime | int |
| max_playtime | int |
| min_age | int |
| game_category | text |
| game_mechanic | text |
| expansion | text |
| game_family | text |

Created *bgg_items* table (366,735 records):

| Fields: | Types & Constraints: |
|---|---|
| game_id | bigint primary key |
| name | text |
| type | text |
| weight | numeric(6,5) |
| year_published | int |
| min_players | int |
| max_players | int |
| min_playtime | int |
| max_playtime | int |
| min_age | int |

Created *bgg_mechanics* table (595,953 records). The *bgg_mechanics* table contains one mechanic per line, so game_ids are repeated when they have multiple mechanics. I want to put all of the mechanics into one record with a single game_id in order to join with my other tables later. I will first create a new table with the data from the old table formatted as I need, and then drop the old table.

**PostgreSQL:**

```
CREATE TABLE temp_mech AS
SELECT game_id, STRING_AGG(mechanics,', ') AS mechanics
FROM bgg_mechanics
GROUP BY game_id
ORDER BY game_id;
```

```
DROP TABLE IF EXISTS bgg_mechanics;

ALTER TABLE temp_mech RENAME TO bgg_mechanics;
```

The final *bgg_mechanics* table (366,735 records):

| Fields: | Types & Constraints: |
|---|---|
| game_id | bigint |
| mechanics | text |

For my project I need to combine data from all 4 tables into a new. Some of the datasets are more updated than the others, but not all of them contain the same data, so I will pick and choose from what I have available. A lot of the data needs proper rounding or renaming, so all of that will be done in one query.

**PostgreSQL:**

```
CREATE TABLE bgg_games AS
SELECT r.game_id, r.name, r.year_published, r.rank, ROUND(r.rating,1) AS rating, r.users_rated,
ROUND(i.weight,2) AS weight, r.abstracts_rank,r.cgs_rank, r.childgames_rank, r.familygames_rank,
r.partygames_rank, r.stratgames_rank, r.thematic_rank, r.wargames_rank, i.min_players AS min_player,
i.max_players AS max_player, i.min_playtime, i.max_playtime, i.min_age, d.game_category, m.mechanics AS
game_mechanic, d.expansion, d.game_family
FROM bgg_ranks AS r
LEFT JOIN bgg_items AS i USING (game_id)
LEFT JOIN bgg_details AS d USING (game_id)
LEFT JOIN bgg_mechanics m USING (game_id)
WHERE r.users_rated >= 1729
ORDER BY r.game_id;
```

# Narrowing Down the Data

The site offered data on over 300,000 titles! I realized this also included a lot of things like game expansions and video games, so I decided to filter those out and stick with just titles classified as “board games,” which gave me 151,199 titles to work with. These titles may be originals, or stand alone expansions of another game. It's still a lot of titles, but I need to check the data. A lot of them can be self-published niche games that only have 1 user rating. I'm looking for the most popular games, but I also don't know yet how to define “popular” for this project. I will start with some aggregations to see what I am dealing with.

Every title has a 'users_rated' field to show us how many people left a rating for it. A person picks a rating of 1-10 and then those numbers are averaged; a very standard ratings system.

Taking a look at the median users_rated, my query returned a resounding 4! Out of 151,199 titles, half of them have 4 or less ratings. I can't consider those titles if I'm looking for the most popular games, but how many ratings does a title need to be considered “popular,” or at least known?

The mean of users_rated is 182, significantly higher than our median. Well, how does this compare with the title that has the most ratings? The title with the most user ratings is CATAN, with 124,451. That's still a long way from 182. So, what number is the cut off? 1,000? 10,000?

There are 4,166 games that have at least 1,000 user ratings. That is a very small number compared to my dataset of 151,199 titles. The Board Game Geek website runs its own Bayesian average of 30 ratings on games to prevent manipulation of the rankings. Does that mean 31 ratings is the number to consider a game as well-known?

Looking at the population standard deviation of the user ratings, we get 1,729. If a game received at least 1,729 rating on the website, then I can assume there is an even larger population of players which have not rated the game. This is the number I will use as the basis for further insights. I will assume that the games which exist in this category at least have a following large enough that not every fan of the game knows each other. That will be my definition of “popular” for this project. This gives us 2,667 titles to work with.

My final *board_games* table for visualizations contains 2,667 records:

| Fields: | Types & Constraints: |
|---|---|
| game_id | bigint primary key |
| name | text |
| year_published | int |
| rank | int |
| rating | numeric |
| users_rated | int |
| weight | numeric |
| abstracts_rank | int |
| cgs_rank | int |
| childgames_rank | int |
| familygames_rank | int |
| partygames_rank | int |
| stratgames_rank | int |
| thematic_rank | int |
| wargames_rank | int |
| min_player | int |
| max_player | int |
| min_playtime | int |
| max_playtime | int |
| min_age | int |
| game_category | text |
| game_mechanic | text |
| expansion | text |
| game_family | text |