Creating a Movie Review Database Pipeline

Kevin Havis

2024-09-06

Creating a Movie Review Database Pipeline

Today we'll be building a low-cost database that can dynamically handle movie reviews that have been submitted to our company, Bruised Potatoes, which is well known for publishing fair and honest takes from critics and fans alike.

Reviewers will submit their reviews via our movie survey Google Form. These collected responses cover reviews from our valued critics on some of the most popular movies in recent years, rating them from "Great" to "Bad".

To get a better sense of our critic's tastes, we've also invited them to offer their favorite movie.

Our database will composed of three tables;

1. The `movies` table, which will contain information about specific movies. This will allows us to add new movies in the future
2. The `reviewers` table, which will contain information about our critics, such as name and favorite movie
3. The `reviews` table, which contains both the movies, the reviewers, and their reviews

Loading the survey data

We could access the Google Sheet directly, but for convenience we've extracted the results into the "movie_survey_responses.csv" file.

```
install.packages('tidyverse')
```

```
movies <- read_csv("movie_survey_responses.csv", show_col_types = FALSE)

glimpse(movies)
```

```
## Rows: 5
## Columns: 11
## $ Timestamp                                <chr> "8/31/2024 21:23:30", "8/31~
## $ 'Alien: Romulus'                         <chr> "Have not seen", "Have not ~
## $ 'The Barbie Movie'                       <chr> "Have not seen", "Have not ~
## $ 'Dune: Part Two'                         <chr> "Have not seen", "Have not ~
## $ Oppenheimer                              <chr> "Great", "Have not seen", "~
## $ 'Poor Things'                            <chr> "Have not seen", "Have not ~
## $ 'Killers of the Flower Moon'             <chr> "Have not seen", "Have not ~
```

```
## $ `Asteroid City`                              <chr> "Have not seen", "Have not ~
## $ `Deadpool & Wolverine`                       <chr> "Have not seen", "Have not ~
## $ `First Name`                                 <chr> "Keith", "Corey", "Jeff", "~
## $ `What is your favorite movie of all time?` <chr> "Lord of the Rings", "Airpl~
```

Unfortunately the formatting Google has stored the reviews in is not ideal for a SQL database. The movies are stored as columns along with the reviewer names and their favorite movies, and the actual scores are stored as values.

We will need to do some data transformation in order to allow this data to sit neatly in our three tables. Let's start with the `movies` table

The movies table

```
# Drop time stamp column and clean column names
movies <- movies |>
  select(everything(), -Timestamp) |>
  rename('reviewer_name' = 'First Name', 'favorite_movie' = 'What is your favorite movie of all time?')
```

```
# Extract movies into a new tibble
movie_table <- tibble(names(movies)) |>
  rename('film' = 'names(movies)') |>
  filter(film != 'reviewer_name' & film != 'favorite_movie')

glimpse(movie_table)
```

```
## Rows: 8
## Columns: 1
## $ film <chr> "Alien: Romulus", "The Barbie Movie", "Dune: Part Two", "Oppenhei~
```

We've extracted the reviewed movies, but we'd also like to add our reviewer's favorite movies.

```
# Extract favorite movies and append to movie_table
fav_movies <- movies |>
  select(favorite_movie) |>
  filter(!is.na(favorite_movie)) |>
  rename('film' = 'favorite_movie') # allows us to join with movie_table

# Append favorite movies to the movie table
movie_table <- bind_rows(movie_table, fav_movies)

glimpse(movie_table)
```

```
## Rows: 13
## Columns: 1
## $ film <chr> "Alien: Romulus", "The Barbie Movie", "Dune: Part Two", "Oppenhei~
```

Now we have our complete and well-structured `movies` table. We'll add this to our database later. Let's continue on with our remaining two tables.

The reviews table

We've asked reviewers to submit their reviews as a qualitative measure, such as "Poor" or "Good". We'll keep these values, but we should also add a corresponding numerical value. This will allow us to calculate metrics down the road.

```
# Reformat the reviews table
reviews_table <- movies |>
  pivot_longer(cols =(0:8), names_to = 'film', values_to = 'rating') |>
  select(everything(), -'favorite_movie')

# Add numerical rating field
reviews_table <- reviews_table |>
  mutate(num_rating = case_when(
    rating == 'Great' ~ 5,
    rating == 'Good' ~ 4,
    rating == 'Okay' ~ 3,
    rating == 'Poor' ~ 2,
    rating == 'Bad' ~ 1,
    TRUE ~ NA
  ))
```

The reviews table is done! Let's move on to our last table; the reviewers themselves.

The reviewers table

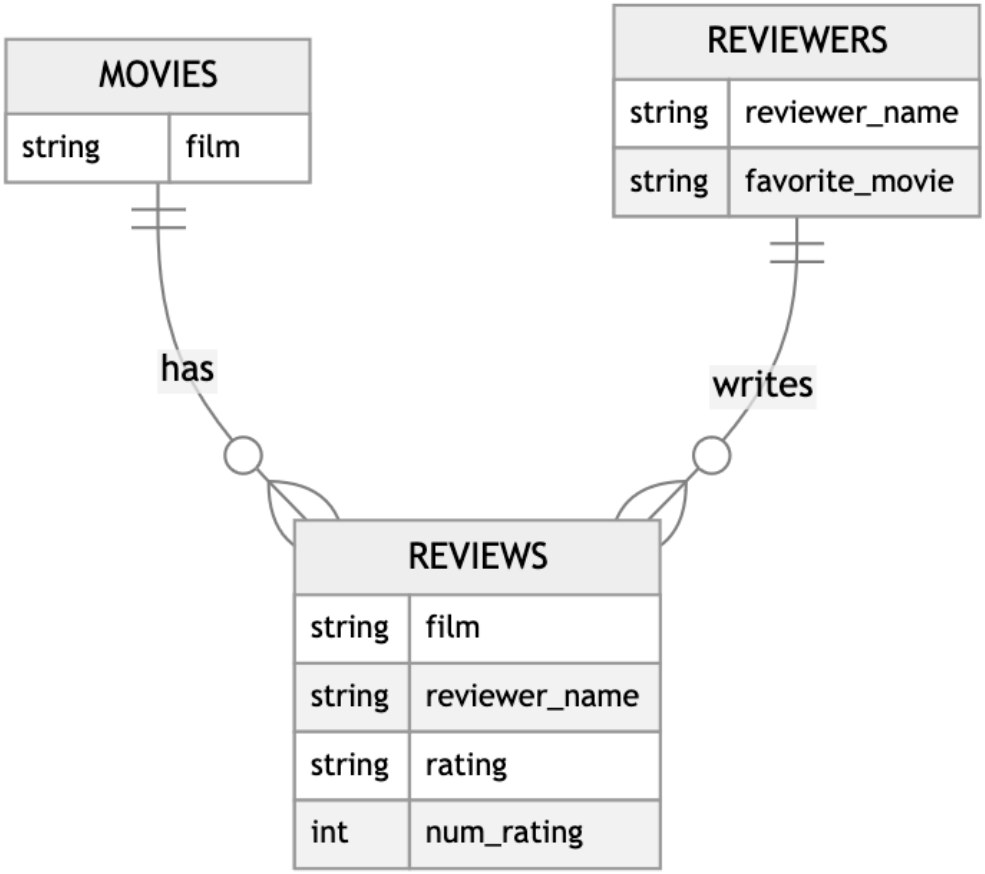
As mentioned before, we'd like to capture both the reviewer and their favorite movie. This gives us a better understanding of each reviewer's unique tastes, which could be useful if we later choose to develop a recommender service; fans who have similar tastes to a critic's will be recommended movies the critic enjoys more often.

```
# Get reviewer names
reviewer_table <- movies |>
  select(reviewer_name, favorite_movie)
```

Building the database

So far we've been able to dynamically format and clean the data into three datasets, but they are only in memory. We will now build a locally hosted database and add these tables.

Designing the database to have three tables allows us to manage relationships between the tables.




```
# Initialize the database connection
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = "bruised_potatoes.db")
```

```
# Write our in-memory datasets to the actual database tables
dbWriteTable(con, "movies", movie_table, overwrite = TRUE)

dbWriteTable(con, "reviews", reviews_table, overwrite = TRUE)

dbWriteTable(con, "reviewers", reviewer_table, overwrite = TRUE)
```

```
# Review our databases tables
dbListTables(con)
```

```
## [1] "movies"    "reviewers" "reviews"
```

Our database has been properly set up and can be appropriately scaled, queried, an even uploaded to a cloud database easily.

Using our new data asset

Now that we have this data in a proper database, we can ask interesting questions about our reviewers and their preferences.

What are our reviewer's tendencies in review scores?

```
tbl(con, sql(
  '
  SELECT reviewer_name, MEAN(num_rating)
  FROM reviews
  GROUP BY reviewer_name
  '
))
```

```
## # Source:   SQL [5 x 2]
## # Database: DuckDB v1.0.0 [root@Darwin 23.6.0:R 4.4.1//Users/kevinhav/projects/D607_assign_2/bruised
##   reviewer_name `mean(num_rating)`
##   <chr>                      <dbl>
## 1 Corey                     NA
## 2 Jeff                       4.14
## 3 Taylor                     3.5
## 4 Sean                       3
## 5 Keith                      5
```

Interestingly, we can see that Corey has not seen any of these movies, and Keith has a very generous average rating of 5 / 5!

Let's see what Keith's movie preferences are by checking their record in the `reviewers` table.

```
tbl(con, sql(
  "
  SELECT *
  FROM reviewers
  WHERE reviewer_name = 'Keith'
  "
))
```

```
## # Source:   SQL [1 x 2]
## # Database: DuckDB v1.0.0 [root@Darwin 23.6.0:R 4.4.1//Users/kevinhav/projects/D607_assign_2/bruised
##   reviewer_name favorite_movie
##   <chr>         <chr>
## 1 Keith         Lord of the Rings
```

We'll assume that's the Lord of the Rings trilogy as a collective; it's hard to choose one.

Do we see them rating recent fantasy-adjacent movies highly?

```
tbl(con, sql(
  "
  SELECT *
  FROM reviews
  WHERE reviewer_name = 'Keith'
  "
))
```

```
## # Source:   SQL [8 x 4]
```

```
## # Database: DuckDB v1.0.0 [root@Darwin 23.6.0:R 4.4.1//Users/kevinhav/projects/D607_assign_2/bruised_
##   reviewer_name film                        rating        num_rating
##   <chr>         <chr>                       <chr>              <dbl>
## 1 Keith         Alien: Romulus              Have not seen         NA
## 2 Keith         The Barbie Movie            Have not seen         NA
## 3 Keith         Dune: Part Two              Have not seen         NA
## 4 Keith         Oppenheimer                 Great                  5
## 5 Keith         Poor Things                 Have not seen         NA
## 6 Keith         Killers of the Flower Moon Have not seen         NA
## 7 Keith         Asteroid City               Have not seen         NA
## 8 Keith         Deadpool & Wolverine        Have not seen         NA
```

Interestingly, Keith has only reviewed Oppenheimer, a historical-event based drama on the creation of the atom bomb, and quite different from Lord of the Rings. We can see now why his mean rating was 5 - it was the only rating.

```
dbDisconnect(con)
```

Conclusion

In this article, we created an end-to-end process to store movie reviews in our Bruised Potatoes database. We took the below steps;

- Designed a survey for movie critics
- Built a dynamic and low-cost intake solution via Google Forms
- Performed data transformation to standardize and enhance our data to make it more scalable and discoverable
- Built a working database prototype with appropriate design and schemas
- Used our new data asset to find interesting patterns in our critic's movie habits

Special thanks to those who submitted surveys!

Future iterations

Some ideas for future iterations of this project;

- Add additional features to our tables, such as producers, actors, genres, etc.
- Migrate to a more robust DBMS solution to allow for more complex schemas and data
- Incorporate cloud based services to allow for scalability
- With sufficient data, build a recommender system to suggest like-minded critics and movies to audience members