DS-7330 – File Org & DBMS Term Project – Movies Insights

Ravi Sivaraman, David Grijvala, and Apurv Mittal, SMU – MSDS – DBMS Class

**Abstract**— This paper is to publish the results of a project to build a RDBMS database to be able to store and query the movie datasets to answers the user’s movie trivia.

**Index Terms**—Term Project, Movies Insights, RDBMS.

————————————————

1. Ravi Sivaraman is a student of Data Science with Southern Methodist University, Dallas, TX. E-mail: [rsivaraman@smu.edu](mailto:rsivaraman@smu.edu).
2. David Grijalva is a student of Data Science with Southern Methodist University, Dallas, TX. E-mail: [dgrijalva@smu.edu](mailto:dgrijalva@smu.edu).
3. Apurv Mittal is a student of Data Science with Southern Methodist University, Dallas, TX. E-mail: [apurvm@smu.edu](mailto:apurvm@smu.edu).

—————————— ◆ ——————————

# 1 Introduction

W

e all watch movies, and it has become part of our pop-culture. There have been times when our curiosity goes beyond the showbiz we see on the screen, we want to learn more about the movie like: who directed *Avatar*? how much money did *Top Gun* make, how many successful movies *Tom Hanks* has made, did *Tom Hanks* and *Spielberg* ever work together?

There are lots of website available to satisfy some of our curiosity but somehow, they are limited by what you can query them all at one place. We decided to take data from two most popular movie databases and join them together to build an efficient database which can help answer some of these questions.

# 2 Methodology

To enable the capability of investigating movie insights to answer the questions of interest, we created a database, views and queries to generate desired reports.

## 2.1 Data Identification and Source

There are multiple sources available to capture the information about the movies. We decided to take data from two popular moview websites IMDB.Com and BoxOfficeMojo.com [1],[2]. IMDB publishes the exhaustive data about the movies which includes details about the movie title, languages, cast & crew, IMDB ratings etc. However, this dataset doesn’t include information about the boxoffice collections, MPAA ratings, production house etc.

To capture the missing information in the IMDB dataset we used a published dataset on Kaggle by Igor Kirko [3]. This dataset is screen scrapped information from BoxOfficeMojo.com and includes details of movie budjet, worldwide collections, MPAA ratings and production house from 1990 to 2020.

## 2.2 Entity- Relationship Diagram

We created an Entity-Relationship diagram to build the schema for the proposed movies database. This required the database to be in 3NF (Third Normal Form).

Following relations were created:

Movies\_CC 🡪 Cast & crew linked to a movie

Cast\_Crew 🡪 Personal details of cast & crew

CC\_Profession 🡪 Cast & crew profession(s)

Movie 🡪 Details about the movie

CC\_KnownFor 🡪 Cast & Crew’s popular movie titles

Movie\_Genre 🡪 Genre of the movie

Genre 🡪 List of possible genres

Diagram

Description automatically generated

Fig. 1. Entity Relationship Diagram

## 2.3 Data Extraction

The dataset from IMDB contains 542,000 movie records and 9.2 M cast & crew information spread across seven files. We filtered the dataset for the movies released in US between 1980 to 2020.

We used Python and R program to extract the data from tsv (tab separated values) files, filter and join among different datasets.

The dataset required to be parsed because the information captured from IMDB datasets were not in 1NF (First Normal Form). Once the data was parsed and joined it was extracted into csv files ready to be uploaded in the database.

## Data Wrangling

TSV (Tab separated files) downloaded from IMDB.com and Kaggle.com.

These TSV files are stored as tab separated file, and they are described as above. The database schema is different compared to the TSV files. These files must be stored in the table schema. We use R lang and Linux shell scripts to format the data as required.

R-lang is used to load the files and then create a data frame as the table columns and remove other unnecessary columns and save as CSV file.

#Read files

movies\_crew=read.csv(‘crew.tsv’, sep="\t", header=TRUE)

movie\_title = read.csv(‘title.tsv’,sep = '\t',header = TRUE )

movie\_boxoffice = read.csv(‘boxoffice.csv’, header = TRUE)

movie\_basics = read.csv(‘basics.tsv’,sep = '\t',header = TRUE )

movie\_ratings = read.csv(‘ratings.tsv’,sep = '\t', header = TRUE )

movie\_principals = read.csv(‘principals.tsv’,sep = '\t',header = TRUE )

#Select only columns needed for DB Table

db\_basics = movie\_basics %>% filter(endYear == "\\N") %>% select("titleId", "primaryTitle", "isAdult", "startYear", "runtimeMinutes")

#Remove TV episodes which are not null for end year (Shows concluded)

db\_basics <- movie\_basics %>% filter(endYear == "\\N")

db\_movie <- merge(db\_title, db\_basics, by("titleId")

write.csv(db\_movie, file="db\_movie.csv", row.names=FALSE)

The parsed csv file was used to upload the file into table using “LOAD DATA” command of SQL.

use movies;

LOAD DATA INFILE '/var/mysql\_secure\_file/db\_movie.csv'

INTO TABLE movie

FIELDS TERMINATED BY ','

ENCLOSED BY '"'

LINES TERMINATED BY '\n'

IGNORE 1 LINES

# Queries

We build SQL queries to extract interesting insights from the database. This required table joins to get the desired output.

Here is a sample list of out of the box queries built which will be used to display the results on the front-end application.

1. List top 10 movies in terms of gross revenue
2. Billion Dollar Movies
3. First Billion Dollar Movie in Revenue
4. First Billion Dollar Movie in Profit
5. Movies with the Biggest Budget
6. Movies with the Biggest Losses
7. Movies <celebrity> known for
8. Movie Title and its Genre
9. Average Revenue by Genre
10. Average Ratings by Genre
11. Highest Rated Movies

Sample Queries:

#Movies <celebrity> known for

SELECT

cast\_crew.name Name, movie.title Movie,

Movie.Year Year, Movie.Distributor

'Production House', Movie.worldwide\_boc

Collections, cc\_profession.profession

Profession

FROM

cast\_crew,

cc\_knownfor,

movie,

cc\_profession

WHERE

cast\_crew.name like '%cruise%'

AND

cc\_knownfor.cast\_crew\_id =

cast\_crew.id

AND

cc\_profession.cast\_crew\_id =

cast\_crew.id

AND

movie.id = cc\_knownfor.movie\_id;

# Movie Title and its Genre

SELECT

M.ID,

M.Title 'Movie Title',

Distributor 'Production House',

MG.Genre\_ID,

G.Type 'Genre',

M.worldwide\_boc Revenue

FROM

movie M

INNER JOIN

(Movie\_Genre MG

INNER JOIN Genre G

ON MG.Genre\_ID = G.ID)

ON

M.ID = MG.movie\_id

ORDER BY M.ID;

# Web Application

We built a Front-End application “Movie Insights” and deployed it on the Google Cloud to allow users to access the application from anywhere.

The application lets users select one of many out-of-the-box queries and view the results. Users can easily customize the queries by configuring selected parameters. Application also allows users to build their own queries from scatch.

The results are displayed in the tabular form along with graphs to highlight the interesting facts. Users have the ability to view the results in list or graphical form.

The application has two main components, a server and a front end. The server is built in python using the Flask micro framework. The server is connected to the database by using the **pymsql** library as the connector. Each request sent to the server opens a new connection to the database and performs the requested query.

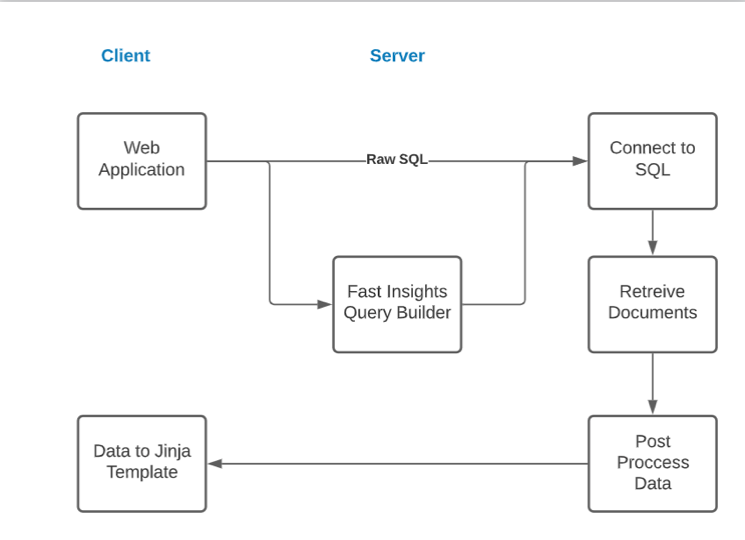


Fig. 2. Client Server Architecture

## Server

To support the application requirements, the application can take two type of SQL queries.

1. A raw SQL query, which allows advance users build their own queries.
2. ***Quick Insights*** are out-of-the-box pre-made queries, which are easy to use and understand. For example: List of top movies a celebrity is known for, allows users to enter the name of the celebrity, specify how many results to display and also to sort the data.

The point of the quick insights is to democratize SQL so that users can use it without any knowledge of data or SQL. This functionality provides 5 scenarios that empower the user to craft their own queries and satisfy their curiosity.

In the backend, the quick insight is an f-string with variable placeholders which are set once the user selects an option on the front end. Once the user has selected all the fields and send the form, the database connection is established, and the f-string is passed as the SQL query.

# Sample Code

# f-string for Query-Movies <celebrity> known for

query = f"""  
 SELECT DISTINCT  
 M.title Movie, M.Year  
 FROM  
 movie M,  
 Cast\_Crew CC  
 WHERE  
 M.ID IN (SELECT   
 C.movie\_id  
 FROM  
 Cast\_Crew CC  
 INNER JOIN  
 CC\_knownfor C ON cc.id = c.cast\_crew\_id  
 WHERE  
 cc.id = (SELECT   
 ID  
 FROM  
 Cast\_crew  
 WHERE  
 Name LIKE '%{name}%'))

ORDER BY year {sql\_sort}  
 LIMIT {sql\_limit};  
 """

## Client

The front end is built using the Jinga templating library, HTML and CSS for styling. The front end is design in such a way it does not only displays the data returned from the SQL query, but it also makes building a query extremely easy with the ***Quick Insights*** functionality. The form layout makes filling out the data extremely easy and intuitive. We decided to humanize this action by making it similar to Graphical user interface, website

Description automatically generatedwriting a sentence.

Fig. 3. Screenshot of the Quick Insights Front End

# Cloud Architecture

Graphical user interface, application, Teams

Description automatically generated

# Reports

The front-end website is installed on Google Cloud and can be accessed at this address: http://35.197.4.53:5000/

The results from the query requested by the user are displayed on the UI in a tabular format. Depending upon th query executed, the columns and rows of the relation can vary.

Below is an example from the query to list the movies actor *Leonardo DiCaprio* is known for:

Graphical user interface, application, Teams

Description automatically generated

Fig. 6. Screenshot of the Reports from the Front End

# Conclusion

As discussed above, this project utilized the publicly available data about the movies. At first the Relational Database was setup with normalized schema. To import the data in the database, ETL (Extract Transform and Load) standard procedure was used.

First, the data was Extracted from the public databases. Second, the data was Transformed which included clean-up, filtering and transformation from tab separated values to relational values using Python. Third and final step was to upload the data to the Relational Database designed using R and Linux scripts.

Then the queries were developed to extract the information from the data.

Finally, the database was connected to the front-end UI hosted on google cloud to present the queries and findings easily accessible to the end users.

The project can be accessed at this web address: http://35.197.4.53:5000/.

**Acknowledgment**

The authors wish to thank Dr. Sohail Rafiqui for his guidance through the duration of this course and the project.

**References**

1. IMDB Dataset downloaded from <https://www.imdb.com/interfaces/>
2. Box Office data captured from <https://www.boxofficemojo.com>
3. Box office data screen scrapped and parsed by Igor Kirko published at <https://www.kaggle.com/igorkirko/wwwboxofficemojocom-movies-with-budget-listed?select=Mojo_budget_update.csv>