The IMDB and Oscar Award OLAP Application Analysis

Roan Campo
De La Salle University
Manila, Philippines
roan_campo@dlsu.edu.ph

Renzo Chua
De La Salle University
Manila, Philippines
renzo_chua@dlsu.edu.ph

Kenneth Go
De La Salle University
Manila, Philippines
kenneth_dy_go@dlsu.edu.ph

Nathaniel Adiong
De La Salle University
Manila, Philippines
nathaniel_irvin_adiong@dlsu.edu.ph

Keywords

Data Warehouse, ETL, OLAP, Query Processing, Query Optimization

1 Introduction

The IMDB dataset includes information about different films and series across time. In this project the group included an external dataset called Oscar Awards from Kaggle [2] to include the awards given to different crew members including directors ,actors, writers and more. With that in mind the group designed the data warehouse to centralize over the relationship of films, crew members and awards over time. The OLAP (Online Analytical Processing) application on the other hand tackles the analysis of these relations and the target audience for this application are for stakeholders who are involved in the film industry. These stakeholders can be venture capitalist [1], directors, producers and writers.

2 Data Warehouse

The data warehouse design was designed in mind with the Oscar Awards dataset found in Kaggle to analyze the performance of titles over time for business intelligence.

2.1 Starflake Schema

The group decided to use the Starflake schema, a mix of star and snowflake to get the most out of both schemas by having some of the tables denormalized while maintaining a hierarchical structure. Not only that but the group also factored out the numerous joins required given the number of dimensions the original source dataset has, combining them into a denormalized dimension table became the better option. A good example of this was when the group was deciding on how to tackle genre. Given that the original source data from *title.basics* had a column genre with more than one entry seperated by column if the group uses Star Schema the data will be too bloated while Snowflake Schema will make joins more costly due to requiring a bridge table for Genres to Titles.

2.2 Fact Tables

2.2.1 Oscar Awards. To better integrate the Kaggle Dataset for Oscar Awards the group created a fact table *OscarAwards* which is tied to *DimPerson* and DimTitle. The data inside the data set follows the same format of IMDB where the format contains tt and nm followed by a list of numbers for the title and nominee respectively. In the table the data follows columns *canonical_category*, *category*,

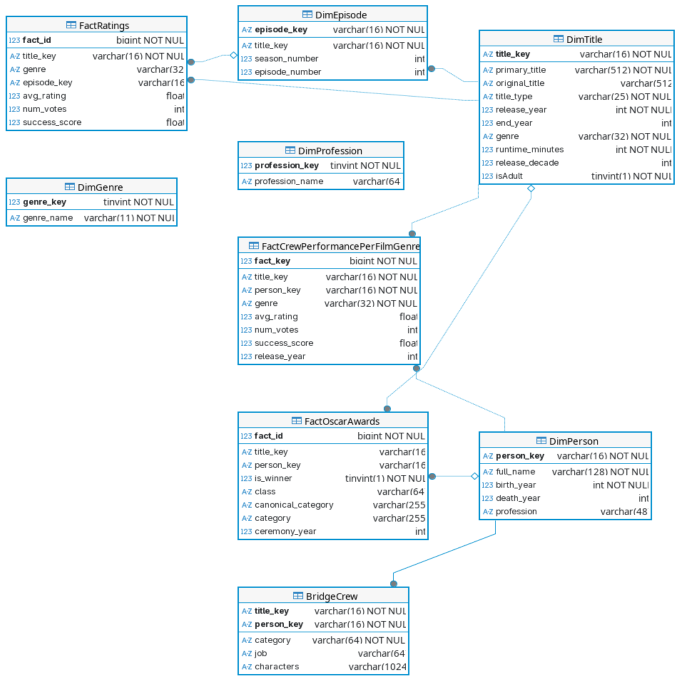

FactRatings
fact_id bigint NOT NUL
title_key varchar(16) NOT NUL
genre varchar(32
episode_key varchar(16
avg_rating float
num_votes int
success_score float
DimEpisode
episode_key varchar(16) NOT NUL
title_key varchar(16) NOT NUL
season_number int
episode_number int
DimTitle
title_key varchar(16) NOT NUL
primary_title varchar(512) NOT NUL
original_title varchar(512
title_type varchar(25) NOT NUL
release_year int NOT NULI
end_year int
genre varchar(32) NOT NUL
runtime_minutes int NOT NULI
release_decade int
isAdult tinyint(1) NOT NUL
DimProfession
profession_key tinyint NOT NUL
profession_name varchar(64
DimGenre
genre_key tinyint NOT NUL
genre_name varchar(11) NOT NUI
FactCrewPerformancePerFilmGenre
fact_key bigint NOT NUL
title_key varchar(16) NOT NUL
person_key varchar(16) NOT NUL
genre varchar(32) NOT NUL
avg_rating float
num_votes int
success_score float
release_year int
FactOscarAwards
fact_id bigint NOT NUL
title_key varchar(16
person_key varchar(16
is_winner tinyint(1) NOT NUL
class varchar(64
canonical_category varchar(255
category varchar(255
ceremony_year int
DimPerson
person_key varchar(16) NOT NUL
full_name varchar(128) NOT NUL
birth_year int NOT NULI
death_year int
profession varchar(48
BridgeCrew
title_key varchar(16) NOT NUL
person_key varchar(16) NOT NUL
category varchar(64) NOT NUL
job varchar(64
characters varchar(1024

Figure 1: Starflake Schema

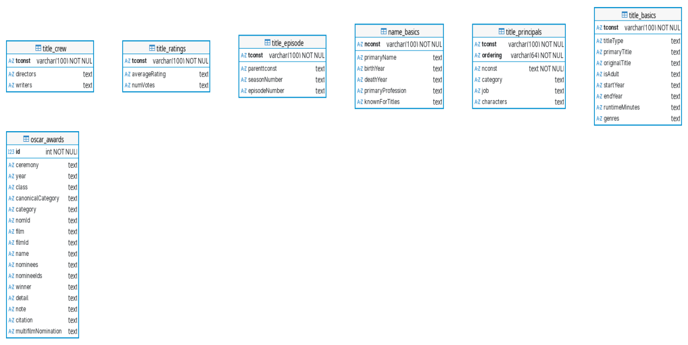


Figure 2: Source Schema

class which ties to what category the nominee is involved in where *canonical_category* gives the more specific category for their nomination while *class* gives a more broad idea for the category. The *release_year* in this table shows the time dimension on when the award was given.

2.2.2 *Ratings.*

On the other hand FactRatings gets the ratings data from the original source which includes the series key and episode key if its a series else it outputs only the film key where episode key is null if its not contained in the episode dimension table. The table includes a one hot encoded genre string which will be explained more in detail in Section 3

2.2.3 *Crew Performance Per Film Genre.*

Lastly the table FactCrewPerformancePerGenre stores the crew performance given the performance of the film or episode and the entire crew for that film. Each row contains one crew member tied to one film or episode and the rating details from FactRatings. The table also includes a column *success_score* while will be explained more in detail in Section 4.

2.3 Dimensions

In the schema there are six dimension tables, DimEpisode, DimTitle, DimPerson, DimProfession, DimGenre, and BridgeCrew

2.3.1 *Title.*

DimTitle stores the title data from the *title.basics* and the year data of each title.

$$ReleaseYear \rightarrow ReleaseDecade$$

The hierarchy goes from Year to Release Decade where Release Decade is a derived column. The table also stores a one hot encoded version of the genre. Besides this the dimension table also stores the release and end years as well as the titles name and a boolean to know if its an adult film or not. The purpose of having all these is to support Slice and Dice operations.

2.3.2 *Episode.*

DimEpisode stores the episode data containing the season number and episode number as the hierarchy.

$$Title \rightarrow SeasonNumber \rightarrow EpisodeNumber$$

2.3.3 *Person.*

DimPerson contains the full name of the person, birth year and a death year if its included. The DimPerson table also includes a one hot encoded version of the profession given that it is possible for the person to have more than 1 profession.

2.3.4 *Genre.*

DimGenre is a lookup table that contains the indexes and name of the genre for the one hot encoded genre in the previous tables. The main reason the group decided to stick to one hot encoding was because it became easier to store the genres given a title since the original dataset gave a list of genres per title.

2.3.5 *Professions.*

DimProfession is a lookup table that contains the indexes and name of the profession as well. The same reasoning applies to the one hot encoded version of professions due to it being easier to read and analyze a one hot encoded string rather than a string seperated by comma for the professions.

2.3.6 *Crew.*

BridgeCrew contains the category type of job and the job of the person given the title. The dimension table also has an optional characters field given that the person was an actor for that title. This table also includes the hierarchy of category and job as followed where category is the job category and job being the specific role.

$$Category \rightarrow Job$$

2.4 Issues Encountered

2.4.1 *Denormalizing Genre and Profession.*

One of the original issues that was addressed by the one hot encoding were the constant fields that could be one hot encoded such as genre and professions since all it takes is to check whether or not the title applies it. This is because if the group converts it to bridge tables for professions and crews respectively the cost of doing joins will be too high. Besides this the group created functions in MySQL to convert the input genres seeprated by comma to one hot encoded data.

2.4.2 *Many to Many Relationships.*

During the ETL one of the problems that the group encountered was converting data from lists from the original dataset to multiple rows. For example to convert the directors and writers from title crews to BridgeCrew the group had to flatten the strings and map it out to rows. To solve this problem the group decided to use a recursive common table expression to map out the dataset to BridgeCrew.

3 ETL Script

3.1 Extracting

The total storage size of the IMDB datasets and the Oscar Awards dataset we used amounts to around 6.8GB totaling around 150 million rows across all datasets. The group decided to drop the *title.akas* dataset since it is expensive to operate with many strings across many languages, and the values of the dataset are not useful for the purposes of our OLAP application.

The group chose to use all of the remaining datasets and extracted only the columns that will be relevant for the OLAP application. The IMDB datasets provide relevant information regarding the titles and genres of films, the crew and their roles, the persons, and the ratings of a certain film. Additionally, the group chose to use an Oscar Awards dataset from Kaggle to extract the films or persons who were nominated. These datasets have the purpose of gauging the popularity of films and the persons involved with the film.

3.2 Transforming

For the transformation process, the group took into account null and mismatched values and transformed them accordingly. For null values, it will be set to a default value or kept as null depending on the constraint of the data warehouse table. The group decided to use the *INSERT IGNORE INTO* command for some of the extraction queries to disallow the script from adding mismatched values since only a few rows have mismatched values and all edge cases cannot be easily dealt with.

Multiple columns in the datasets contain string array values separated by commas that the group needed to account for. For string arrays that have a fixed amount of values such as genres and professions, the group decided to use one-hot encoding to query the values quicker without having to join with another table or generate more rows for each value. Rather than having multiple columns that corresponds to each value, which would take up too much space, the one-hot encoding is formed from a fixed length string, where each character in the string corresponds to a genre or profession with a **T** or **F** value, indicating true or false respectively. This way, the database can maintain the one-hot encoding while the backend of the application can interpret the value. To execute this in SQL, the group created a function that loops across the comma separated string, getting each value using the **STRING_INDEX** function and concatenating a **T** or **F** value to the one-hot encoding string. The result of the string will then be added to a column on their respective tables.

For string arrays that do not have a fixed amount of values, such as the person keys from the crews and Oscars table, each person will be separated and have a separate row to allow for querying the foreign key values. This is done by using the *RECURSIVE* statement to iterate over each value in the comma separated string, where each value is gotten from the **SUBSTRING_INDEX** function. Each value gotten from the comma separated string will then be unioned then inserted to its respective table.

Since the fact tables were made with faster querying in mind, then some of its values have to be queried from the other tables in the data warehouse, leading to a potentially longer execution time.

The group added foreign key constraints to various keys to maintain data integrity, thus the group also used the *INSERT IGNORE INTO* command to the queries to disallow values that don't follow the constraints from being added.

3.3 Loading

Initially, the group wanted to load the datasets into the data warehouse using Python, however its operations proved to be too slow and complicated to load the large amount of data in a reasonable amount of time. For example to load the original dataset into a source table the time it took to load the dataset was around seven hours. The group decided instead on using pure SQL to load the datasets to a source MySQL database which took two hours.

Additionally, a major problem the group encountered during the loading process was accounting for null and mismatched values. Across the various datasets, there we many empty values even on common values like names and titles. There were also values that are
N, which also indicates that there is no value for that cell. There were mismatched and uncleaned values along the datasets such as having the value *Reality-TV* be on the *runtimeMinutes* column rather than the *genres* column. To make up for all different values, the datatypes of all source tables are set to *TEXT*, making the actual database after transfer take up two times more storage than the total storage across the datasets.

The ETL process takes a long time to execute due to the sheer amount of data across all datasets that the script needs to go through. As a result, the group ran into issues when trying to execute the script. Our database is hosted in a server, so if the server goes down for any reason, the ETL script would stop. The group tried tackling these issues by making the overall process faster, like shifting from Python to SQL to faster querying and using a computer with a stronger CPU rather than a Raspberry Pi. Besides these optimizations, there was not much the group can do but wait for all the processes to finish.

4 OLAP Application

The **IMDb Analytics Dashboard** is a web-based OLAP application designed to analyze and visualize movie industry data for enhanced decision-making. Built using **Next.js** and **React** for the frontend and **MySQL** for the backend, it provides users such as producers, directors, investors, and studios with interactive tools to explore trends, evaluate performance, and discover meaningful patterns across movies, actors, genres, and awards.

4.1 Main Purpose

The main purpose of the **IMDb Analytics Dashboard** is to support **data-driven decision-making** in the film industry by aggregating, summarizing, and analyzing large sets of movie-related data. Through multi-dimensional OLAP operations such as **roll-up**, **slice**, **dice**, and **drill-down**, the application enables users to view information from different perspectives and at varying levels of detail. This allows stakeholders to identify popular actors, successful genres, professional distributions, and award trends which can help contribute to smarter production, casting, and investment choices.

4.2 Analytical Reports and SQL Implementation

4.2.1 Popular Actors by Success Metric. This subsection explores what are the popular actors based on the generated Success metric. Take note that the Success metric is equal to the following:

$$\textbf{Success} = \textbf{Average Rating} \cdot \log(1 + \textbf{Number of IMDB Votes})$$

The intuition behind this metric is that success of a show should be correlated with two other metrics:

- **Popularity**, represented by Number of IMDB Votes
- **Reception**, represented by the Average Rating.

To illustrate the point, If two shows both have a higher rating, yet one has more votes than the other in IMDB, then the latter show should be considered more successful. Conversely, two shows that have roughly the same votes, yet the first one has better ratings, then the former should be considered more successful.

The Logarithm transformation is there to normalize the Number of IMDB Votes. Moreover, two shows that have high number of votes with a difference in the tens or hundreds should be roughly the same in terms of success.

The following is the given SQL script for Popular Actors by Success Metrics.

```
WITH ActorStats AS (
SELECT
  bc.person_key,
  COUNT(DISTINCT bc.title_key) AS total_titles,
  AVG(fr.success_score) AS avg_rating
  FROM FactRatings fr
JOIN BridgeCrew bc ON fr.title_key = bc.title_key
WHERE bc.category IN ('actor', 'actress')
GROUP BY bc.person_key
)
SELECT
```

```
    dp.full_name,
    a.total_titles,
    a.avg_rating, -- success_score is renamed as avg_rating
        for the app
    RANK() OVER (ORDER BY a.avg_rating DESC, a.total_titles
        DESC) AS actor_rank
FROM ActorStats a
JOIN DimPerson dp ON dp.person_key = a.person_key
LIMIT 10;
```

This operation is a Roll-up for from it aggregates the success metrics titles at the actor-level. With the following query, we obtain the following information:

full name	total titles	avg rating	actor rank
RJ Mitte	1	139.58201599121094	1
Steven Michael Quezada	1	139.58201599121094	1
Emilia Clarke	2	130.2659740447998	3
Peter Young-blood Hills	1	124.63971710205078	4
John Bradley	3	123.87860479915844	5
Noah Schnapp	1	122.23243713378906	6
Joe Keery	1	122.23243713378906	6
Natalia Dyer	1	122.23243713378906	6
Tony Sirico	1	121.51786804199219	9
Cricket Leigh	1	120.3641586303711	10

Table 1: Top 10 Popular Actors ranked by Average Success Score (renamed as avg rating).

The results show that RJ Mitte and Steven Quezada are the top actors based on the metric.

4.2.2 Popular Genres by Success Metric. This section explores what are the popular genres based on the generated Success metric. To reiterate, the success metric is the following:

$$\textbf{Success} = \textbf{Average Rating} \cdot \log(1 + \textbf{Number of IMDB Votes})$$

Based on this metric, we will use the ff. SQL query to answer it.

```
    SELECT
        dt.genre,
        AVG(fr.avg_rating) AS avg_rating,
        AVG(fr.success_score) AS success_score,
        COUNT(DISTINCT dt.title_key) AS total_titles
    FROM FactRatings fr
    JOIN DimTitle dt ON fr.title_key = dt.title_key
    WHERE dt.release_year BETWEEN YEAR(CURDATE()) - 10 AND
        YEAR(CURDATE())
    GROUP BY dt.genre
    ORDER BY success_score DESC, avg_rating DESC
    LIMIT 10;
```

This operation is a Roll-up, for it aggregates from title-level at the genre-level.

These are one hot combinations of genres. To illustrate, "FFFFFFFTFFTFF[illegible]" is a combination of genres which encodes Crime, Drama, and Thriller.

genre	avg rating	success score	total titles
FFFTFTFFF···	8.7	99.4699707031	1
FFTFFFTFF···	8.1	82.9242706299	1
FFFFFFFF···	8.6	76.4835510254	1
FFFFFFFF···	7.9	74.2151870728	1
FFFTFFFF···	7.6	70.323460799	4
FFFFFFFT···	7.5	70.0993440787	75
FFFFFFFT···	7.6	69.4265899658	1
FFTFFFFT···	7.5	68.5701917252	2
FFTFFFFT···	7.6	67.7071075439	1
FFFFFFFF···	8.1	67.6852767657	2

Table 2: Top 10 Popular Genres by Success Metric.

4.2.3 Popular Movies of a Given Name by Success Metric. Here, we try to answer the question what are the popular movies a person's name is related to.

```
WITH PersonInfo AS (
    SELECT person_key
    FROM DimPerson
    WHERE full_name = ?
)
SELECT
    DISTINCT fcp.title_key AS title_key,
    fcp.avg_rating AS avg_rating,
    fcp.num_votes AS num_votes,
    fcp.success_score AS success_score
FROM FactCrewPerformancePerFilmGenre fcp
JOIN PersonInfo pi ON pi.person_key = fcp.person_key
ORDER BY success_score DESC;
```

This operation is a slice, for filters the data by a single actor name parameter. Given that the input parameter for 'full_name' is "Robert Downey Jr.", we obtain the following data:

title key	avg rating	num votes	success score
tt14404618	6.8	9046	61.9493
tt3473134	8.4	343	49.0614
tt8421554	7.8	488	48.3004
tt20297790	6.3	437	38.318
tt0390776	6.5	209	34.7562
tt1618221	5.6	86	25.0091
tt0827928	8.4	17	24.2791
tt2354581	7.8	18	22.9666
tt32159454	6.3	29	21.4275
tt10334498	5.0	70	21.3134

Table 3: Top 10 Popular Movies of "Robert Downey Jr."

The title key for the top movie for this actor is "The Sympathizer", with an average rating of 6.8, but a whopping 9046 number of votes.

4.2.4 Top Oscars Awards by Canonical Category. This subsection explores the what are the top canonical category with the most Oscar awards.

The following is the SQL statement:

```
WITH TopCanonicalCategories AS (
   SELECT
   foa.canonical_category AS canonical_category
   FROM FactOscarAwards foa
   WHERE foa.is_winner = 1
)
SELECT
   canonical_category,
   COUNT(*) AS total_wins
FROM TopCanonicalCategories
GROUP BY canonical_category
ORDER BY total_wins DESC
LIMIT 10;
```

This operation is a Roll-up, for it aggregates the awards from category to canonical category; Moreover, this can be considered a Drill-down, for it is going from category to canonical category. This results with the following results:

Canonical Category	**Total Wins**
SCIENTIFIC AND TECHNICAL AWARD (Technical Achievement Award)	348
SCIENTIFIC AND TECHNICAL AWARD (Scientific and Engineering Award)	250
VISUAL EFFECTS	232
SOUND MIXING	205
MUSIC (Original Song)	176
ART DIRECTION	160
BEST PICTURE	146
DOCUMENTARY (Feature)	136
WRITING (Adapted Screenplay)	135
HONORARY AWARD	126

The results show that the canonical category "SCIENTIFIC AND TECHNICAL AWARD (Technical Achievement Award)" has the most total wins with 348 wins. Followed by "SCIENTIFIC AND TECHNICAL AWARD (Scientific and Engineering Award)" with 250 wins. This shows that progress in the science and technical field of entertainment is highly valued to getting an Oscar Award.

4.3 Visualized EDA

4.3.1 Ratio of Professions of Crew Members. This section explores the ratio of professions of crew members represented in a Pie Chart.

```
SELECT
   bc.category AS profession,
   COUNT(*) AS count
FROM BridgeCrew bc
WHERE bc.category IS NOT NULL
GROUP BY bc.category
ORDER BY count DESC
LIMIT 10;
```

This is a roll-up OLAP operation, for it aggregates crew members at the category level. With this query, the following information is obtained.

profession	count
actor	20660285
actress	16211647
self	13091534
writer	12117519
director	8984456
producer	5151458
editor	4071060
cinematographer	3323530
composer	2927487
production designer	1086588

This results shows that the profession with the most number of crew members are actors with 20,660,285, followed by actress with 16,211,647.

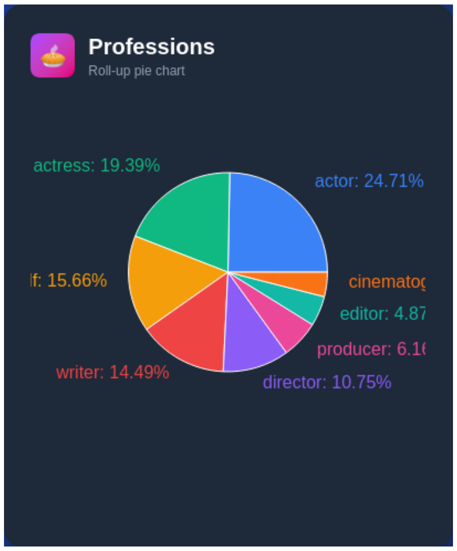


Figure 3

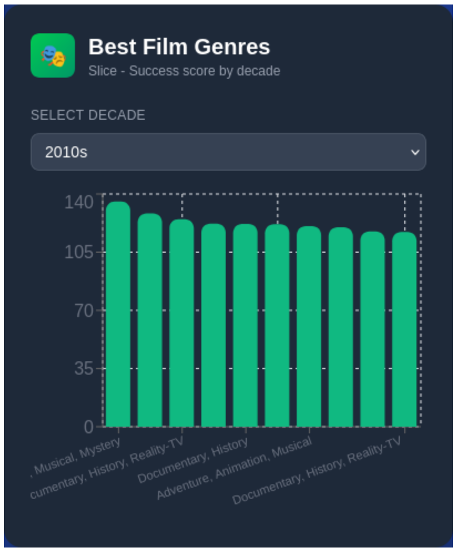
4.3.2 Best Film Genre within the Past Decade. This section explores the best film genre in a Bar Graph.

```
WITH GenreSuccess AS (
SELECT
dt.release_decade AS decade,
dt.genre AS genre,
fr.success_score AS success_score
FROM FactRatings fr
JOIN DimTitle dt ON fr.title_key = dt.title_key
)
SELECT
decade,
genre,
AVG(success_score) AS avg_success_score
FROM GenreSuccess
WHERE decade = $1
GROUP BY decade, genre
ORDER BY avg_success_score DESC
LIMIT 10;
```

This query performs a Dice OLAP operation, for it filters (decade constraint) and groups by genre. simultaneously. Given that we have decade as 2010, we obtain the following information from the query:

decade	genre	success score
2010	FFFTFTFFFFFF· · ·	99.469970703125
2010	FFFTFFFFFFFF· · ·	82.13441467285156
2010	FFFFFFFTFTFF· · ·	78.83134842966939
2010	FFFFFFFTFFTF· · ·	78.17122020721436
2010	FFFFFFFFTFFF· · ·	76.80824279785156
2010	FFFTFFFFFFTF· · ·	75.39966583251953
2010	FFFFFFFFFFFF· · ·	71.67395782470703
2010	FFFFFFFTFFTF· · ·	70.41068196296692
2010	FFFTFFFFFFFF· · ·	69.62734985351562
2010	FFFFFFFTFFFF· · ·	69.42658996582031

The genre here is a one hot encoding of combination genres. Following decoding of the genre, titles with combination of 'Musical' and 'Mystery' genre tends to have a high success score within the 2010s decade. A function can be used to decode what does the genre combination mean



SELECT DECADE
2010s

Figure 4

4.3.3 *Successful movies per Given Genre over Given Decade.*

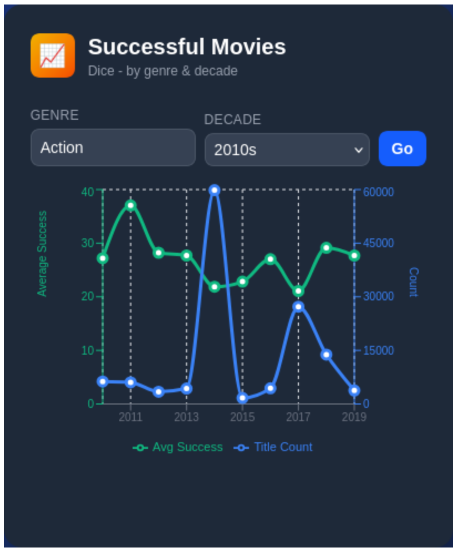
This section explores what are the most successful movies given a combination of genre over a given decade.

```
SELECT
    title_key,
    release_year,
    success_score
FROM FactCrewPerformancePerFilmGenre
WHERE genre = ?
AND release_year BETWEEN ? AND ?
GROUP BY title_key, release_year, success_score
ORDER BY release_year;
```

This operation is a Roll-up and Dice OLAP operation, for it aggregates based title key, release year, and success score, and aggregates based on genre and release year. Given that given genre is "FFFFFFFTFFTFFTFFFFFFFFFFFFFFF" which encodes Crime, Drama, and Thriller; and the release decase is from 2000 to 2010. The results shows the ff.:

title key	release year	success score
tt0383175	2003	36.2522
tt0350456	2003	34.8341
tt0465689	2005	56.1745
tt1047931	2007	69.7203
tt1100911	2008	51.1033
tt1460941	2009	13.0914
tt1321865	2010	73.0037

This genre combination shows the overall trendline for movies success. The title key refers to a movie's orignal title named "L'Affaire Dominici" or its alternative name as "The Dominici Case" in 2003. To illustrate the idea, of how it can be graphed. The following is the scatter plot for Action Movies in the 2010s.



GENRE
Action
DECADE
2010s

Figure 5

4.4 Statistical Tests

4.4.1 *Correlation test with Ratings and Votes.*

This section explores whether or not the number of votes for a given title in the IMDB website has a correlaton with the rating it has.

```
WITH OverallRatings AS (
    SELECT
    AVG(avg_rating) AS overall_avg_rating,
    AVG(num_votes) AS overall_votes
    FROM FactRatings
),
RatingsDifference AS (
    SELECT
    (avg_rating - (SELECT overall_avg_rating FROM
        OverallRatings)) AS ratings_difference,
    (num_votes - (SELECT overall_votes FROM OverallRatings))
        AS votes_difference
    FROM FactRatings
)
SELECT
    (SUM(ratings_difference * votes_difference) /
    (SQRT(SUM(POW(ratings_difference, 2))) *
    SQRT(SUM(POW(votes_difference, 2))))) AS pearson_r
FROM RatingsDifference;
```

Based on the query above, we can conduct the following hypothesis test to determine whether there exists a statistically significant relationship between IMDb ratings and number of votes. The null hypothesis states that there is no correlation between these two variables, while the alternative hypothesis states that a correlation exists.

- H_0: There is **no linear relationship** between ratings and votes ($\rho = 0$)
- H_1: There **is a linear relationship** between ratings and votes ($\rho \neq 0$)

Given the following is the output of the result:

Pearson Correlation
0.06869894408882023

and assuming the number of samples (n) is sufficiently large (i.e., $n > 30$), the test statistic for Pearson's correlation is:

$$t = r\sqrt{\frac{n-2}{1-r^2}}$$

which follows a Student's t-distribution with $(n-2)$ degrees of freedom.

Using the correlation result from the previous SQL query and substituting $r = 0.0687$, the test yields a very small t-statistic, implying a high p-value ($p > 0.05$). Therefore, we **fail to reject the null hypothesis**, meaning there is no statistically significant correlation between IMDb ratings and number of votes.

This is supported by the the fact that if we try to create scatter plot for number of votes and ratings. This graphically does not show a clear linear relationship.

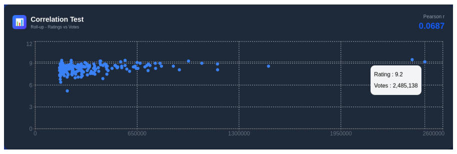


Figure 6: Scatter Plot IMDb Ratings and Votes

Statistic	**Value / Interpretation**
Pearson Correlation (r)	0.0687
t-statistic	$\approx$ 1.25 (for large n)
p-value	> 0.05
Decision	Fail to reject H_0
Interpretation	There is no statistically significant correlation between IMDb ratings and the number of votes. This suggests that the number of votes does not necessarily imply higher ratings.

This test validates that IMDb's weighted rating system may not have a direct linear relationship between the number of votes and the computed average rating. Thus, other factors such as vote weighting, voter credibility, or IMDb's internal normalization may influence the published ratings.

5 Query Processing and Optimization

Query Optimization Overview: Query optimization is used to reduce the amount of time querying data from the database. Query optimization uses multiple SQL query techniques and commands to optimize selecting data from one or more tables and prevent doing too many operations or joining too many rows, leading to a slower return.

5.1 Strategies Applied

- **Indexing**: Created indexes for the fact tables and dimension tables with the ff.:

```
CREATE INDEX idx_factratings_title ON
    FactRatings(title_key);
CREATE INDEX idx_factoscar_person ON
    FactOscarAwards(person_key);
CREATE INDEX idx_ftcgp_genre_year ON
    FactCrewPerformancePerFilmGenre (release_year);
CREATE INDEX idx_bridgecrew_title_category ON
    BridgeCrew(title_key, category);
CREATE INDEX idx_bridgecrew_person ON
    BridgeCrew(person_key);
CREATE INDEX idx_factratings_title_avg ON
    FactRatings(title_key, avg_rating, num_votes);
CREATE INDEX idx_bridgecrew_person_category ON
    BridgeCrew(person_key, category);
CREATE INDEX idx_dimtitle_titlekey ON
    DimTitle(title_key);
CREATE INDEX idx_factoscar_category_winner ON
    FactOscarAwards (class, canonical_category,
    is_winner);
CREATE INDEX idx_factoscar_year ON FactOscarAwards
    (ceremony_year);
CREATE INDEX idx_dimtitle_releaseyear ON
    DimTitle(release_year);
CREATE INDEX idx_factratings_titlekey ON
    FactRatings(title_key);
CREATE INDEX idx_dimtitle_genre_year ON
    DimTitle(genre, release_year);
CREATE INDEX idx_factratings_avgvotes ON
    FactRatings(avg_rating, num_votes);
```

```
CREATE INDEX idx_dimperson_fullname ON
    DimPerson(full_name);
```

- **Query Restructuring**: CTEs and selective joins reduced intermediate result sizes.
- **Materialized Columns**: Success metric as a `GENERATED ALWAYS STORED` column avoids recomputation.
- **Hardware Optimization**: MySQL buffer pool increased from 1GB to 8GB.
- **Recursion Depth Optimization**: MySQL CTE recursion size increased from 1000 to 10000.

5.2 Query-Specific Optimization

5.2.1 *Popular Actors by Success Metric.*

- The query groups performance metrics by `person_key` with a ranking window function.
- Optimization: Index on `BridgeCrew.person_key` and `Fact Ratings. title_key` to reduce join cost.

5.2.2 *Popular Genres by Success Metric.*

- Uses aggregated averages of `success_score` grouped by `genre`.
- Optimization: Indexed `DimTitle.genre` and used CTE to pre-filter release years to reduce scan time.

5.2.3 *Popular Movies of a Given Actor by Success Metric.*

- Joins filtered by a single `person_key` retrieved from `DimPerson`.
- Optimization: Used parameterized CTE to limit the search space.

5.2.4 *Top Oscars Awards by Canonical Category.*

- Aggregates awards grouped by canonical category.
- Optimization: Index on `FactOscarAwards.canonical_category` improves grouping speed.

5.2.5 *Ratio of Professions of Crew Members.*

- A roll-up operation that counts categories.
- Optimization: Added index on `BridgeCrew.category` for faster grouping.

5.2.6 *Best Film Genre within the Past Decade.*

- Pre-aggregated using decade-level roll-ups on `release_year`.
- Optimization: Generated column `release_decade` and indexed it to reduce computation per query.

5.2.7 *Successful Movies per Given Genre over Given Decade.*

- A temporal trend analysis (line graph) query using roll-up and slice.
- Optimization: Added composite index on `(genre, release_year)` to improve scan performance.

5.2.8 *Correlation Test with Ratings and Votes.*

- Computes Pearson correlation from FactRatings.
- Optimization: Derived averages via CTEs to minimize repeated scans and reduced memory usage by precomputing sums.

OLAP Optimization:

- **Roll-up/Drill-down**: Pre-aggregated data in FactCrewPerformancePerFilmGenre to allow queries to move between year and decade levels.
- **Slice/Dice**: Filtering subsets efficiently through indexed columns (e.g., by genre, release decade).

6 Results and Analysis

Functional Testing:

- Verified correctness of ETL loading and derived columns.
- Confirmed that generated columns (e.g., success_score, release_decade) computed accurately.
- Checked validity of OLAP operations (roll-up, slice, and dice) through manual aggregation validation.

Performance Testing: Testing was conducted by running each analytical query under two conditions: the default 1GB MySQL buffer pool and an optimized configuration with 8GB. Execution time was measured across multiple runs using `EXPLAIN ANALYZE` and the MySQL slow query log.

Query Name	**1 GB**	**8 GB**
Popular Actors by Success Metric	6m 11s	1m 42s
Popular Genres by Success Metric	6m 3s	19s
Popular Movies of a Given Name by Success Metric	5m 36s	0.016s
Top Oscars Awards by Canonical Category	2m 31s	0.02s
Ratio of Professions of Crew Members	2m 28s	36s
Best Film Genre within the Past Decade	1m 49s	2s
Successful Movies per Given Genre over Given Decade	5m 24s	2m 45s
Correlation Test with Ratings and Votes	1m 42s	6.7s

Table 4: Performance comparison between default 1GB and optimized 8GB MySQL configurations.

Analysis: Query execution time significantly improved after index creation and increasing available memory. Denormalized structures (fact tables) improved analytical aggregation speed, while dimension indexing reduced join costs. The correlation test query benefited from reduced intermediate CTE recalculation.

Overall, optimization strategies led to up to high reduction in query time for complex analytical workloads.

7 Conclusion

7.1 Project Summary

The **IMDb Analytics Dashboard** project focused on the design and implementation of a web-based OLAP (Online Analytical Processing) application for movie industry analytics. The system was built using **Next.js** and **React** for the frontend and **MySQL** for the backend, integrating data warehousing and multidimensional analysis concepts. It provides users with interactive tools to explore datasets through operations such as **roll-up**, **slice**, **dice**, and **drill-down**. These allow users to analyze movie trends, actor performance, genre popularity, and award statistics in varying levels of detail.

7.2 Learnings and Insights on Database Concepts

- **Importance of Building and Maintaining a Data Warehouse:** A data warehouse serves as a centralized repository for integrating and organizing data from multiple sources. In this project, it provided a consistent and historical dataset that enabled long-term trend analysis, unlike traditional transactional databases that only store current operational data.
- **Role of ETL (Extract, Transform, Load):** The ETL process was essential in cleaning, transforming, and loading IMDb data into structured dimensional tables. This ensured that the warehouse remained accurate and up to date. The extraction phase gathered raw movie data, transformation standardized formats (such as converting genres and professions into dimension tables), and loading populated the analytical schema for OLAP queries.
- **Purpose of OLAP versus OLTP:** Unlike OLTP systems used in daily transactions, OLAP is designed for analytical querying and decision support. OLAP enables summarization, aggregation, and pattern discovery allowing users to gain strategic insights rather than just process records. In this project, OLAP operations enabled exploration of high-level summaries and detailed statistics that would be difficult to perform efficiently in an OLTP setup.
- **Need for Query Optimization:** Query optimization improves performance and responsiveness in analytical applications. Since OLAP queries often involve large aggregations and joins, optimization strategies such as indexing key attributes, and Common Table Expression significantly reduced computation time.
- **Indexing Strategies:** While MySQL automatically generates indexes for primary and foreign keys, additional indexes can be created on frequently filtered attributes such as genre name or actor name to accelerate query execution. Custom indexes are especially useful when dealing with large-scale data or repetitive aggregation queries.

7.3 Contributions and Societal Relevance

The **IMDb Analytics Dashboard** contributes both to end users and to the database development community. For industry stakeholders, it enables informed decision-making by providing visual insights into audience behavior, genre trends, and performance metrics. For database developers, it serves as a practical example of how OLAP principles, data warehousing, and ETL pipelines can be applied to real-world datasets. Beyond its technical aspects, the project promotes the use of data analytics for creativity and efficiency in film production, marketing, and investment decisions.

After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication

8 References

References

[1] Akhilesh Ganti. 2025. *Venture Capitalists: Who Are They and What Do They Do?* https://www.investopedia.com/terms/v/venturecapitalist.asp October 20, 2025.

[2] David Lu Raphael Fontes. 2025. *The Oscar Award, 1927 - 2025.* Retrieved October 18, 2025 from https://www.kaggle.com/datasets/unanimad/the-oscar-award

9 Declarations

During the preparation of this work the author(s) used ChatGPT, Claude to assist with the following tasks:

- Front-end Code for the OLAP.

Starflake Schema was created by Roan Campo, and curated by Renzo Chua and Kenneth Go. ETL Script was done by Renzo Chua and Kenneth Go. Nate Adioing and Roan Campo did OLAP Application. Perfomance Testing was done Renzo Chua and Roan Campo. IT support and Platform Maintance was done by Renzo Chua.