ANIME FILM PULSE

Enhancing Film Industry Insights and Decision-Making

Jagarlamudi Govinda Lahari
govindal@buffalo.edu
Govindal – 50539692

Sumana Madhireddy
sumanama@buffalo.edu
sumanama – 50510465

Gollapudi Venkata Sai Revathi
vgollapu@buffalo.edu
vgollapu - 50537880

Names	*Vgollapu*	*Sumanama*	*Govindal*
Contribution	*33.33%*	*33.33%*	*33.33%*

Abstract: The film industry revolves around the creation, distribution, and enjoyment of movies, but managing the vast amount of crucial information associated with them can be challenging. To simplify this process, we propose the development of a specialized system known as an "anime movie database." Like a well-organized library, all stakeholders involved in movie production can access essential data, such as box office earnings and release dates. By providing timely and comprehensive information, the movie database will empower producers, distributors, marketers, investors, and others to make informed decisions about movies. Additionally, it will enable them to identify emerging trends and opportunities within the industry, thereby enhancing their competitiveness and efficiency. Ultimately, the anime movie database system will serve as an asset for the entire film industry, facilitating collaboration and contributing to the creation of high-quality movies for audiences worldwide.

I. PROBLEM STATEMENT.

Producing movies involves keeping track of lots of details like titles, costs, and how much money they make. But it's not just that; knowing when they come out, how long they are, and how popular the movie got. Imagine if all this info is scattered everywhere, like in different files or even written on paper. It's a mess! But a movie database system fixes that. It gathers up all the details and puts them neatly in one place, like sorting files into folders on your computer. But it's not just about tidying up. Once everything's neat and gathered in one place, it's easy for everyone involved in making movies to find what they need. Producers, marketers, and directors can log in and see what's happening without searching through piles of papers or long lists. And here's the cool part: this system can do more than just store data. It can help analyze things like which kinds of movies people like the most or how much money a new movie might make based on past ones. It's like having a magic tool that helps make better decisions about which movies to make and how to sell them. So, having a great movie database system isn't just about being organized—it's a powerful tool that helps the movie business stay ahead, make better movies, and attract more viewers.

a) Why Database

Data Volume and Complexity: Each movie in the film industry generates a ton of data, including information, production costs, box office receipts, etc. The efficiency with which Excel files can handle datasets of this size and complexity is limited. This data can be stored and managed more efficiently in a database, which can easily handle the volume and complexity of the data.

Relationships: A film consists of several interrelated components, including genres, release dates, languages of the movie, etc. Using Excel to manage these relationships can be laborious and may cause many errors. Structured relationships between various entities can be established with the help of a database, which guarantees the organization and integrity of data.

Scalability and Performance: The amount of data is increasing as the film business develops and makes more films. This scalability may be too much for Excel files to handle, which could cause problems with data management and performance. A database that is built for performance and scalability can handle the growing volume of data and guarantee peak system performance.

Data Preservation: The film database contains vital information about movies, such as financial information, ratings, an overview of the movie, and other information. Regular backups are necessary to protect this important data. A database management system may automate the backup process, guaranteeing that data is consistently stored in safe locations, in contrast to Excel files, which call for human backup methods.

b) Background and Significance:

New films are consistently released on a variety of platforms and genres, making the film industry extremely competitive and dynamic. In this regard, industry players must have access to precise and current movie data to make well-informed decisions about investment and production. To manage their movie-related data, a lot of firms in the film industry currently use fragmented databases or Excel files. However, these conventional approaches frequently lack the capacity for data analysis, scalability, and efficiency. As a result, businesses could find it difficult to learn valuable insights from their data or to access and update information quickly.
We can overcome these constraints and provide industry stakeholders with an effective tool for data administration and analysis by creating a centralized movie database management system. Processes may be streamlined, data accessibility and quality can be increased, and stakeholders can have greater insights into audience preferences, market trends, and revenue possibilities with the use of such a system.

c) Objective

The primary objective of this project is to develop a robust movie database management system that can effectively store, manage, and analyze movie-related data. Specifically, the system aims to address the following questions:

- How can we efficiently store and organize movie data, including attributes such as title, rating, revenue, and genre?
- What insights can be derived from analyzing movie data, such as trends in popularity, revenue generation, and audience preferences?
- How can this database system facilitate informed decision-making processes for various stakeholders in the film industry?

d) *Contribution to the Problem Domain:*

There are several ways in which the suggested movie database management system might make a big difference in the anime movie business.

- *Better Data Management:* Information about movies will be considerably easier to manage with the new movie database system. Everything will be arranged neatly and centrally, much like when you organize all your movie-related files into

distinct folders on your computer. You will save a ton of time with this technique because it eliminates the need for human labor. It will be simple to search for things such as the release date of a film. Choosing which films to make and how to market them will be much easier and more intelligent with this technology, which will also make handling movie data a breeze.

- *Improved Decision Making:* The new method will make it easier for those working in the film industry to decide which films to produce, market, and invest in. It accomplishes this by providing customers with all the pertinent movie information, such as viewership and release dates. They may use this system to see what's going on in the film industry now and make decisions about what films to make, how to market them, and where to put their money to receive the best returns.

- *Insights and analysis:* By carefully examining all the film-related data, the system will assist those working in the film industry in comprehending what is happening. They can use it to identify patterns and trends, such as the kinds of movies that are popular or the direction that the business is taking.

II. TARGET USER.

a) Who will use your database?

The movie database will serve as a unified hub for all stakeholders involved in the world of filmmaking, fostering collaboration, informed decision-making, and creative exploration. Producers and production companies will utilize the database to manage budgets, track production progress, and monitor the status of film projects. Distributors and exhibitors will rely on it to determine optimal release dates, languages, and genres. Marketers and advertisers will leverage the database to identify target audiences, tailor advertising campaigns, and measure audience engagement metrics.

Investors and financial analysts will assess investment opportunities, scrutinize revenue forecasts, and evaluate the financial viability of film ventures. Additionally, researchers and analysts will utilize the database to conduct market studies, analyze industry trends, and generate valuable insights for both academic and corporate purposes. Regulatory authorities will also benefit from the database, using it to ensure compliance with content ratings, copyright regulations, and financial transparency standards. In essence, the database will empower all stakeholders to navigate the complexities of the film industry with efficiency, agility, and creativity.

b) Who will administer the database?

In a real-world setting, a group of specialists within a film production business would probably oversee the movie database, much like an organization would have an IT team to oversee its computer systems. Consider the following scenario involving a major film studio:

Scenario: Picture "Pixar Studios," a sizable anime picture studio renowned for producing hit movies. To manage their new movie database, they choose to form a unique team they call the "Database Team". There are experts on databases and computers on this team. Large decisions about the functionality and purposes of the database are made by them. The database administrators, who function as the database developers maintain everything's functionality, ensure that it is protected from hackers, and address any issues

that may arise. Subsequently, there exist "Data Analysts."

They search through the database to get important information. They search for trends, patterns, and other items that can assist "Pixar Studios" in producing better films or marketing them to the appropriate audiences. Lastly, some provide technical support. When something goes wrong with the database, such as when you can't locate the information you need or something isn't functioning properly, you should contact them. They aid in resolving the issue so that everyone may resume their jobs. This group works together to ensure that the movie database is operating efficiently.

III. ORIGINAL & FINAL SCHEMA

a) *Original Anime Movies table*

ID: Unique identifier for every movie
Title: Anime movie title
Vote_Avg: rating by viewers on an average
Vote_count: number of votes received.
Status: the movie is released or yet to be released.
Release Date: date when the movie was released.
Revenue: generated by movie
Runtime: duration of movie in minutes
Adult: specifies if the movie is intended for adults
Backdrop_path: path to the backdrop image of the movie.
Budget: The production budget of the movie
Homepage_path: movies' official Page path
IMDB_id: Imdb identifier of movie
Original_language: language in which the movie was originally produced.
Original_title: the original title of the movie
Overview: a brief synopsis of the movie
Popularity: score based on every viewer's engagement
Poster_path: poster image path of the movie
Tagline: phrase associated with the movie
Genre: category of movie like comedy, Rom-Com, Violence.
Genre_id: Unique identifier for the genre of the movie.
Production_company: Company that produced movie.
Prod_company_id: Unique identifier for the production company of the movie.
Production_country: the country where the movie was produced.
Spoken_language: languages spoken in the movie.
Spoken_lang_id: Unique identifier for the spoken language(s) used in the movie.

b) *Functional Dependencies*

- ID → Status, Release_date, Runtime, Adult, Backdrop_path, Homepage_path, Imdb_id, Original_language, Popularity, Poster_path, Tagline.
- ID→ Genre_id, Production_company_id, Spoken_language_id
- Genre_id → Genre
- Prod_company_id→Production_company
- Prod_country_id→Production_country
- Spoken_lang_id → Spoken_language
- Imdb_Id → Title, Vote_avg, Vote_count, Adult, Overview, Popularity, Revenue, Budget
- Title → Original_title

c) Boyce- Codd Normal Form Verification

According to the definition of BCNF (Boyce-Codd Normal Form), a functional dependency is in BCNF if, for every determinant (A), the determinant is a super key.

- ID → Status, Release_date, Runtime, Adult, Backdrop_path, Homepage_path, Imdb_id, Original_language, Popularity, Poster_path, Tagline.

Since the determinant ID in this instance uniquely identifies each tuple in the relation, it is a super key. Consequently, BCNF contains the functional dependency. This indicates that the relationship is free of redundancy and irregularities associated with functional dependencies, as well as partial dependencies.

- ID→ Genre_id, Production_company_id, Spoken_language_id

Since ID is a super key and the determinant of all other attributes in the relation, there is a functional dependency in BCNF.

- Spoken_lang_id → Spoken_language

Spoken_lang_id$^+$ = Spoken_lang_id, Spoken_language. So we decompose R into R_1 into{Spoken_lang_id, Spoken_language_name} now, This is in BCNF.

- Genre_id → Genre

Genre_id$^+$ = Genre_id, Genre. This violates BCNF, hence we decompose this into R_2={Genre_id, Genre_name}. Now, in the decomposed relation R_2, Genre_id (alone or along with Genre) serves as a superkey, satisfying the BCNF criteria.

- Prod_compan_id→Production_company

Prod_compan_id$^+$ = Prod_compan_id, Production_company. Since Prod_compan_id does not uniquely identify each tuple in the relation, it is not a superkey by itself in this instance. On the other hand, Prod_compan_id and Production_company together provide a composite superkey, which means that any tuple in the relation can be uniquely identified by the combination of both characteristics. So, we decompose into R_3={Prod_company_id,Production_company_name}

- Prod_country_id→Production_country

Prod_country_id$^+$ = Prod_country_id, Production_country. Since Prod_country_id does not uniquely identify each tuple in the relation, it is not a superkey by itself in this instance. On the other hand, Prod_country_id and Production_country together provide a composite superkey, which means that any tuple in the relation can be uniquely identified by the combination of both characteristics. So, we decompose into R_3={Prod_country_id,Production_country_name}

- Imdb_Id → Title, Vote_avg, Vote_count, Adult, Overview, Popularity, Revenue, Budget

Imdb_Id$^+$ = {Imdb_Id,Title, Vote_avg, Vote_count, Adult, Overview, Popularity}. This violates BCNF. So, we decompose this into R_8={Imdb_Id, Title, Vote_avg, Vote_count, Adult, Overview, Popularity, Revenue, Budget }. Now R_8 is in BCNF.

- Title → Original_title

$Title^+$ = {Title, Original_title}. This violates BCNF, hence we decompose this into R_{10} = {Title, Original_title}. Now, in the decomposed relation R_{10}, Title (alone or along with Original_title) serves as a superkey, satisfying the BCNF criteria.

R_4 after decomposing the functional dependencies, we have R_4={ID, title, Status, Release_date, Runtime, Backdrop_path, Homepage_path,Imdb_id,Original,Languag,Poster_path, Tagline, Production_company}
Also,Production_company_name,genre_name, spoken_language_name have multi values, to satisfy 1NF, we have decomposed this into R_1, R_2, R_3.

d) Final Schema with BCNF relations

Table 1: Spoken_language_name

Attribute 1: Spoken_lang_id
Description: Unique identifier for each spoken language.
Purpose: Identifying spoken languages uniquely.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Spoken_language_name
Description: Name of the spoken language.
Purpose: Describing the language spoken.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

PK: Spoken_lang_id
Relation schema:
Spoken_language_name(Spoken_lang_id, Spoken_language_name)

Table 2: Genre_name

Attribute 1: Genre_id
Description: Unique identifier for each genre.
Purpose: Identifying genres uniquely.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Genre_name
Description: Name of the genre.
Purpose: Describing the genre.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

PK: Genre_id
Relation Schema:
Genre_name(Genre_id, Genre_name)

Table 3: Production_company_name

Attribute 1: Production_company_id
Description: Unique identifier for each production company.
Purpose: Identifying production companies uniquely.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Production_company_name
Description: Name of the production company.
Purpose: Describing the production company.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

PK: Production_company_id
Relation Schema:
Production_company_name(Production_company_id, Production_company_name)

Table 4: Movie

Attribute 1: ID
Description: Unique identifier for each movie.
Purpose: Uniquely identifying each movie.
Datatype: Integer.

Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Status
Description: Current status of the movie.
Purpose: Indicating the current state.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 3: Release_date
Description: Date when the movie was released.
Purpose: Providing release information.
Datatype: Date or Datetime.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 4: Runtime
Description: Duration of the movie.
Purpose: Informing about the movie's length.
Datatype: Integer or Float.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 5: Backdrop_path
Description: Path to the backdrop image.
Purpose: Displaying backdrop image.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 6: Homepage_path
Description: Path to the movie's homepage.
Purpose: Provide a link to the official homepage.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 7: Imdb_id
Description: IMDb identifier for the movie.
Purpose: Linking to IMDb for more information.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable (Foreign Key).

Attribute 8: Original_language
Description: Language in which the movie was originally made.
Purpose: Specifying the original language.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 9: Poster_path
Description: Path to the movie's poster image.
Purpose: Displaying poster image.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 10: Tagline
Description: Catchphrase or slogan of the movie.
Purpose: Highlighting a memorable line.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

PK: ID
FK: Imdb_id
Relation Schema:
Movie(<u>ID</u>, title, Status, Release_date, Runtime, Backdrop_path, Homepage_path, Imdb_id, Original,Language, Poster_path, Tagline).

Referential integrity constraint: ON DELETE SET NULL (Set the foreign key columns in the referencing table to NULL.)

Table 5: Spoken_language
Attribute 1: ID

Description: Unique identifier for each record.
Purpose: Uniquely identifying each record.
Datatype: Integer.
Default Value: Not applicable.

Nullability: Not nullable (Primary Key).

Attribute 2: Spoken_language_id
Description: Identifier for spoken language.
Purpose: Associating with a spoken language.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Nullable (Foreign Key).

PK: ID, Spoken_language_id
FK: Spoken_language_id
Relation Schema:
Spoken_language(ID, Spoken_language_id)

Referential integrity constraint: ON DELETE SET NULL (Set the foreign key columns in the referencing table to NULL.)

Table 6: Genre

Attribute 1: ID
Description: Unique identifier for each record.
Purpose: Uniquely identifying each record.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Genre_id
Description: Identifier for movie genre.
Purpose: Associating with a genre.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Nullable (Foreign Key).

PK: ID, Genre_id
FK: Genre_id
Relation Schema:
Genre(ID, Genre_id)

Referential integrity constraint: ON DELETE SET NULL (Set the foreign key columns in the referencing table to NULL.)

Table 7: Production_company
Attribute 1: ID
Description: Unique identifier for each record.
Purpose: Uniquely identifying each record.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Production_company_id
Description: Identifier for the production company.
Purpose: Associating with a production company.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Nullable (Foreign Key).

PK: ID, Production_company_id
FK: Production_company_id
Relation Schema:
Production_company(ID,Production_company_id)

Referential integrity constraint: ON DELETE SET NULL (Set the foreign key columns in the referencing table to NULL.)

Table 8: Imdb

Attribute 1: Imdb_Id
Description: Unique identifier for IMDb entry.
Purpose: Uniquely identifying each IMDb entry.
Datatype: String.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Title
Description: Title of the movie.
Purpose: Associating IMDb entry with movie.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable (Foreign Key).

Attribute 3: Vote_avg
Description: Average rating given by users.
Purpose: Providing rating information.

Datatype: Float.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 4: Vote_count
Description: Total number of votes received.
Purpose: Quantifying user engagement.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 5: Adult
Description: Indicates if the content is for adults.
Purpose: Identifying adult content.
Datatype: Boolean or String (with "True" or "False").
Default Value: Not applicable.
Nullability: Nullable.

Attribute 6: Overview
Description: Summary or description.
Purpose: Provide an overview of the movie.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 7: Popularity
Description: Popularity score or rank.
Purpose: Indicating the popularity of the movie.
Datatype: Float or Integer.
Default Value: Not applicable.
Nullability: Nullable.

Attribute 8: Revenue
Description: Total revenue generated by the movie.
Purpose: Tracking financial performance.
Datatype: Numeric (e.g., Integer or Float).
Default Value: Not applicable.
Nullability: Nullable.

Attribute 3: Budget
Description: Budget allocated for the movie.
Purpose: Indicating the cost of production.
Datatype: Numeric (e.g., Integer or Float).
Default Value: Not applicable.
Nullability: Nullable.

PK: Imdb_id
FK: Title
Relation Schema:
Imdb(Imdb_Id, Title, Vote_avg, Vote_count, Adult, Overview, Popularity)

Referential integrity constraint: ON DELETE SET NULL (Set the foreign key columns in the referencing table to NULL.)

Table 9: Production Country
Attribute 1: ID

Unique identifier for each record.
Purpose: Uniquely identifying each record.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: production_country_id
Description: Uniquely identify different production countries associated with each movie.
Purpose: Establish a relationship between the movie and the country where it was produced.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Nullable.

PK: ID, production_country_id
FK: production_country_id
Relation Schema:
Production_Country(ID,Production_country_id)

Table 10: Production_Country_name

Attribute 1: Production_Country_id
Description: Uniquely identify production countries associated with each movie.

Purpose: Establishes a relationship between the movie and the country where it was produced.
Datatype: Integer.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Production_Country_name
Description: stores the full names of the production countries.
Purpose: Provide a standardized list of production country names, linked to their respective IDs.
Datatype: String.
Default Value: Not applicable.
Nullability: nullable

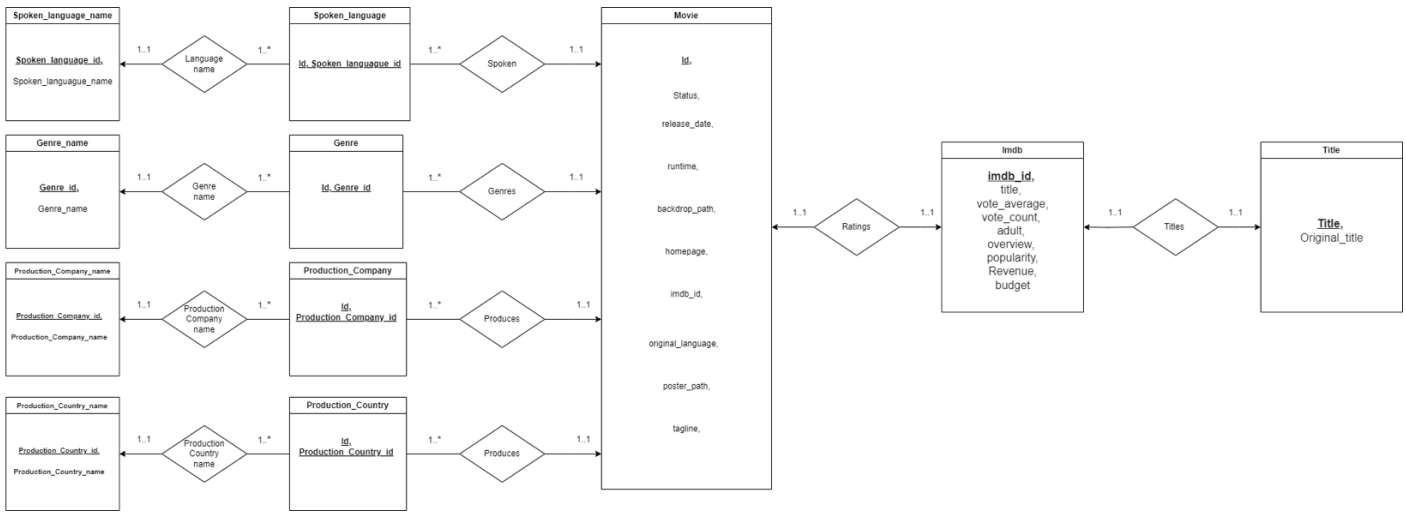
PK: Production_country_id
Relation Schema:
Production_Country_name(Production_country_id, Production_country_name)

Attribute 1: Title
Description: Title of the movie.
Purpose: Identifying the movie by name.
Datatype: String.
Default Value: Not applicable.
Nullability: Not nullable (Primary Key).

Attribute 2: Original_title
Description: Original title of the movie.
Purpose: Provide the title in its original language.
Datatype: String.
Default Value: Not applicable.
Nullability: Nullable.

PK: Title
Relation Schema:
Title(Title, Original_title)

Table 11: Title
ER Diagram:



IV. CHALLENGES WITH LARGE DATASETS.

Query:

Query | Query History

```
1 SELECT *
2 FROM Movie
3 WHERE Runtime > (SELECT AVG(Runtime) FROM Movie);
```

Explain:

Data Output | Messages | Explain × | Notifications

Graphical | Analysis | Statistics

#	Node
1.	→ Seq Scan on movie as movie Filter: (runtime > $0)
2.	→ Aggregate
3.	→ Seq Scan on movie as movie_1

The initial query suffers from poor performance due to executing a subquery for every row in the Movie table to compute the average runtime. This approach becomes particularly inefficient with larger datasets. The execution plan reveals that PostgreSQL is conducting a sequential scan (Seq Scan) on the Movie table and filtering rows based on the condition Runtime > $0, where $0 represents the outcome of the subquery.

Optimization:

CREATE INDEX idx_runtime ON Movie (Runtime); To further optimize the query, we can create an index on the Runtime column to speed up the filtering process.

Optimized Query:

Query | Query History

```
1 WITH avg_runtime AS (
2 SELECT AVG(Runtime) AS avg_runtime
3 FROM Movie
4 )
5 SELECT *
6 FROM Movie
7 CROSS JOIN avg_runtime
8 WHERE Runtime > avg_runtime.avg_runtime;
```

Explain:

	QUERY PLAN text
1	Nested Loop (cost=216.70..367.22 rows=1667 width=195)
2	-> Aggregate (cost=175.50..175.51 rows=1 width=8)
3	-> Seq Scan on movie movie_1 (cost=0.00..163.00 rows=5000 width=8)
4	-> Bitmap Heap Scan on movie (cost=41.20..175.04 rows=1667 width=187)
5	Recheck Cond: (runtime > (avg(movie_1.runtime)))
6	-> Bitmap Index Scan on idx_runtime (cost=0.00..40.78 rows=1667 width...
7	Index Cond: (runtime > (avg(movie_1.runtime)))

The revised execution plan demonstrates enhancements, indicating that the index on the Runtime column (idx_runtime) is being efficiently utilized.

- Initially, the query planner computes the average runtime through an Aggregate node.
- Following this, it conducts an Index Only Scan on the Movie table (movie_1) using the idx_runtime index to retrieve runtime values with efficiency.
- Subsequently, a Bitmap Heap Scan is performed on the Movie table to filter rows where the runtime exceeds the calculated average runtime.
- During the Bitmap Heap Scan, the idx_runtime index is utilized effectively to verify the condition runtime > (avg(movie_1.runtime)).

The query's performance has been enhanced by leveraging the index on the Runtime column, which eliminates the need for sequential scans and allows for efficient filtering of rows based on the runtime condition.

V. DATABASE TESTING: *SQL QUERIES AND EXECUTION RESULTS*

a. Insertion Queries

→ *Query1: Inserting 2 tuples into Genre_ name*

```
Query  Query History
1 INSERT INTO Genre_name (Genre_id, Genre_name)
2 VALUES (19, 'Romance Comedy'), (20, 'Slice of Life');
```

This query introduces two fresh genres, 'Romance Comedy' and 'Slice of Life', into the "Genre_name" table, assigning a unique identification number (Genre_id) to each. Specifically, 'Romance Comedy' is allocated ID 19, while 'Slice of Life' receives ID 20. Upon execution, this query enriches the "Genre_name" table with two additional rows, representing each genre alongside its respective identification number as shown below.

14	13	Romance
15	14	ScienceFiction
16	15	TVMovie
17	16	Thriller
18	17	War
19	18	Western
20	19	Romance Comedy
21	20	Slice of Life

→ *Query2: Inserting two tuples into Spoken_language_name*

```
Query  Query History
1 INSERT INTO Spoken_language_name (Spoken_lang_id, Spoken_language_name)
2 VALUES (59, 'Telugu'), (60, 'Kannada');
```

This SQL query inserts new records into the “Spoken_language_name” table. In simpler terms, imagine this table as a directory listing various languages spoken in movies, such as English, Spanish, or French. With this query, we’re instructing the database to add two new languages: ‘Telugu’ and ‘Kannada’. Each language is assigned a unique identification number (Spoken_lang_id), with ‘Telugu’ being given ID 59 and ‘Kannada’ receiving ID 60. So, when this query is executed, the “Spoken_language_name” table expands by two rows—one for each newly added language—displaying their names alongside their respective identification numbers as shown below.

57	56	Ukrainian
58	57	Yiddish
59	58	Zulu
60	59	Telugu
61	60	Kannada

b. Deletion Queries

→ *Query1: Delete Row with Minimum Vote Count from IMDB table.*

```
Query  Query History
1 DELETE FROM imdb
2 WHERE vote_count = (SELECT MIN(vote_count) FROM imdb);
```

The query targeted rows with a vote count equal to the minimum vote count found in the entire dataset. By executing this query, we aimed to identify and potentially remove entries with the lowest levels of popularity or engagement, as reflected by their vote counts, potentially removing entries that are less popular or have received fewer votes compared to others in the dataset.

→ *Query2: Popularity-Based Deletion from imbd table.*

```
Query  Query History
1 DELETE FROM imdb
2 WHERE popularity BETWEEN 50 AND 51;
3
```

By executing this query, the intention is to remove records with a moderate level of popularity, falling within the specified range. This means that movies with a popularity score between 50 and 51 will be excluded from the dataset, effectively removing entries that are moderately popular compared to others, and potentially refining the dataset to highlight movies with higher or lower levels of popularity.

c. *Updating Queries*

→ *Query1: updating production_country_name*

```
Query  Query History
1 UPDATE public.production_country_name SET production_country_name= 'United States of America'
2 WHERE production_country_name = 'UnitedStatesofAmerica';
```

This SQL query updates the entries in the "production_country_name" table by searching for any instances of 'UnitedStatesofAmerica' in the table and replace them with 'United States of America'. This modification ensures consistency and accuracy in the representation of the United States as a production country, potentially improving data clarity and usability. The comparisons of the country name before and after running the query are shown below.

Before Updating:

65	64	UnitedArabEmirates
66	65	UnitedKingdom
67	66	UnitedStatesofAmerica
68	67	Uruguay
69	68	Yugoslavia

After Updating:

65	64	UnitedArabEmirates
66	65	UnitedKingdom
67	67	Uruguay
68	68	Yugoslavia
69	66	United States of America

→ *Query2: updating language Korean to Hangul in spoken_language_name table.*

```
Query  Query History
1 update public.spoken_language_name set spoken_language_name = 'Hangul'
2 where spoken_language_name = 'Korean'
```

This SQL query specifically targets the "spoken_language_name" table within the "public" schema. In movie databases, the "spoken_language_name" table typically stores information about languages spoken in films. When this query is executed, it updates any occurrences of 'Korean' in the "spoken_language_name" column to 'Hangul'. This update operation ensures that all references to the Korean language are standardized to its native name, 'Hangul', within the dataset. By maintaining consistency in language representation, the query enhances the clarity and integrity of the dataset, facilitating more accurate data analysis and interpretation. Changes are compared below.

Before Updating:

	spoken_lang_id [PK] integer	spoken_language_name character varying (255)
31	30	Italian
32	31	Japanese
33	32	Korean

After Updating:

57	57	Yiddish
58	58	Zulu
59	59	Telugu
60	60	Kannada
61	32	Hangul

d. *Select Queries*

→ *Query 1: Production Company Movie Count Analysis.*

```
Query  Query History
1 SELECT pc.Production_company_name, COUNT(m.ID) AS Movie_Count
2 FROM Production_company_name pc
3 LEFT JOIN Production_company p ON pc.Production_company_id = p.Production_company_id
4 LEFT JOIN Movie m ON p.ID = m.ID
5 GROUP BY pc.Production_company_name
6 HAVING COUNT(m.ID) > 100;
```

This SQL query retrieves the names of production companies along with the count of movies associated with each company from the database. Specifically, it selects the "Production_company_name" column from the "Production_company_name" table and calculates the number of movies associated with each company.

The query achieves this by performing a LEFT JOIN operation between the "Production_company_name" table and the "Production_company" table on their corresponding IDs, and then another LEFT JOIN operation between the "Production_company" table and the "Movie" table based on their respective IDs.

By grouping the results based on the production company name and applying a condition using the HAVING clause to filter out results where the count of associated movies is greater than 100, the query ensures that only production companies with more than 100 movies are included in the result set. This enables users to identify production companies with a significant volume of movie productions in the dataset.

Retrieved Information:

Data Output | Messages | Explain | Notifi

	production_company_name character varying (255)	movie_count bigint
1	WarnerBros.Pictures	246
2	WarnerBros.Animation	213
3	WarnerBros.Cartoons	209
4	WaltDisneyPictures	142
5	Metro-Goldwyn-Mayer	160
6	TOHO	113
7	MGMCartoonStudio	141
8	WaltDisneyProductions	462
9	ToeiAnimation	127
10	Pixar	114

→ *Query 2: Movie Genre Average Runtime Analysis.*

Query | Query History

```
SELECT Genre_name.Genre_name, AVG(Movie.Runtime) AS Average_Runtime
FROM Movie
INNER JOIN Genre ON Movie.ID = Genre.ID
INNER JOIN Genre_name ON Genre.Genre_id = Genre_name.Genre_id
GROUP BY Genre_name.Genre_name
ORDER BY Average_Runtime DESC;
```

The average runtime of each genre of movies in the database is retrieved by this SQL query, which then displays the results in descending order of average runtime. The three tables "Movie", "Genre", and "Genre_name" are joined to accomplish this. It establishes the relationship between movies and genres by first linking the "Movie" and "Genre" tables based on their shared ID column. Then, to retrieve the genre names, it further links the "Genre" and "Genre_name" tables using the genre ID. The average runtime for each genre is then determined by grouping the results of the query based on the genre name. Users can now identify genres with longer average movie runtimes at the top of the list by sorting the results in descending order based on the average runtime.

Retrieved Information:

	genre_name character varying (255)	average_runtime double precision
1	Adventure	77.9625850340136
2	Crime	77.63076923076923
3	Action	75.06390532544378
4	Thriller	74.93975903614458
5	History	72.33333333333333
6	Mystery	71.855
7	Fantasy	71.67179980750721
8	ScienceFiction	70.96111975116641
9	Drama	68.51388888888889
10	Romance	68.1875
11	War	60.91379310344828
12	Family	57.254518808011724
13	TVMovie	56.45933014354067
14	Documentary	55.07142857142857
15	Horror	53.70857142857143
16	Animation	50.34606921384277
17	Music	47.204188481675395
18	Comedy	45.604320078546884
19	Western	36.371428571428574

→ *Query 3: Filtered Movie Data Retrieval*

Query | Query History

```
SELECT Movie.ID, Movie.Runtime, Movie.Release_date, Genre_name.Genre_name,
Production_company_name.Production_company_name
FROM Movie
INNER JOIN Genre ON Movie.ID = Genre.ID
INNER JOIN Genre_name ON Genre.Genre_id = Genre_name.Genre_id
INNER JOIN Production_company ON Movie.ID = Production_company.ID
INNER JOIN Production_company_name ON Production_company.Production_company_id =
Production_company_name.Production_company_id
WHERE Movie.Release_date >= '2023-01-01'
AND Movie.Runtime >= 120
ORDER BY Movie.Release_date DESC;
```

This SQL query retrieves specific details like movie ID, runtime, release date, genre name, and production company name from the "Movie" table. It selects only movies with a runtime of at least 120 minutes and a release

date on or after January 1, 2023. To achieve this, it connects various tables like "Genre", "Genre_name", "Production_company", and "Production_company_name" using their IDs.

This establishes links between movies, genres, and production companies, enabling the extraction of relevant information. The query arranges the results based on the movies' release dates in descending order. Essentially, it helps gather comprehensive data about recent movies meeting specific criteria, such as runtime and release date, alongside their associated genres and production companies.

Retrieved Information:

	id integer	runtime double precision	release_date date	genre_name character varying (255)	production_company_name character varying (255)
1	569094	140	2023-05-31	Action	AradProductions
2	569094	140	2023-05-31	Adventure	AradProductions
3	569094	140	2023-05-31	Animation	AradProductions
4	569094	140	2023-05-31	Action	ColumbiaPictures
5	569094	140	2023-05-31	Adventure	ColumbiaPictures
6	569094	140	2023-05-31	Animation	ColumbiaPictures
7	569094	140	2023-05-31	Action	LordMiller
8	569094	140	2023-05-31	Adventure	LordMiller
9	569094	140	2023-05-31	Animation	LordMiller
10	569094	140	2023-05-31	Action	PascalPictures
11	569094	140	2023-05-31	Adventure	PascalPictures
12	569094	140	2023-05-31	Animation	PascalPictures
13	569094	140	2023-05-31	Action	SonyPicturesAnimation
14	569094	140	2023-05-31	Adventure	SonyPicturesAnimation
15	569094	140	2023-05-31	Animation	SonyPicturesAnimation

→ *Query 4: Movie Data Retrieval for Lengthy Films.*

Query | Query History

```
SELECT Imdb_id, Release_date, Tagline
FROM Movie
WHERE Release_date IN (
SELECT Release_date
FROM (
SELECT DISTINCT Release_date
FROM Movie
WHERE Runtime >= 120
) AS Subquery
ORDER BY Release_date DESC
LIMIT 100
);
```

This SQL query selects the IMDb ID, release date, and tagline of movies from the "Movie" table. It filters the results to include only movies with a runtime of at least 120 minutes. Then, it sorts the release dates of these movies in descending order and selects the top 100 distinct release dates. Finally, it retrieves the IMDb ID, release date, and tagline of movies released on these selected dates. In essence, the query retrieves information about movies with longer runtimes and provides their IMDb ID, release date, and tagline for further analysis.

Retrieved Information:

	imdb_id character varying (255)	release_date date	tagline character varying (255)
2	tt0119698	1997-07-12	The Fate Of The World Rests On The Courage Of One Warrior.
3	tt9362722	2023-05-31	Its how you wear the mask that matters.
4	tt0092067	1986-08-02	One day, a girl came down from the sky…
5	tt0094625	1988-07-16	Neo-Tokyo is about to E.X.P.L.O.D.E.
6	tt5323662	2016-09-17	Sometimes the answer is as simple as learning to listen.
7	tt0032455	1940-11-13	The most sensational sound youll ever see!
8	tt2013293	2013-07-20	We must live.
9	tt10682000	2019-09-27	[null]
10	tt2576852	2013-11-23	A princesss crime and punishment.

VI. QUERY EXECUTION ANALYSIS

→ *Query1:*

Query | Query History

```
SELECT m.status,
COUNT(DISTINCT g.genre_id) AS Num_Genres,
COUNT(DISTINCT p.Production_company_id) AS Num_Production_Companies
FROM Movie m
JOIN Genre g ON m.ID = g.ID
JOIN Production_company p ON m.ID = p.ID
GROUP BY m.status;
```

The problem with this query arises from its approach to counting the number of distinct genres and production companies associated with each movie. In this query, the Movie table is joined with the Genre and Production_company tables based on their IDs. However, this may lead to inflated counts because it fails to consider the many-to-many relationships between movies and genres, as well as movies and production companies. As a result, multiple matches in the joins can cause inaccuracies in the counts.

Unproblematic Query:

```
--unproblamatic query
SELECT m.status,
  (SELECT COUNT(*) FROM Genre g WHERE g.ID = m.ID) AS Num_Genres,
  (SELECT COUNT(*) FROM Production_company p WHERE p.ID = m.ID) AS Num_Production_Companies
FROM Movie m;
```

The unproblematic query resolves this issue by using subqueries to count the occurrences

of genres and production companies directly associated with each movie. Instead of joining tables and counting distinct values, it individually counts the occurrences of genres and production companies where their IDs match the movie's ID. This approach ensures accurate counts by directly considering the relationships between movies and their associated genres and production companies, thus providing a more reliable result.

→ *Query 2:*

```
--problamatic query
SELECT g.Genre_id, s.Spoken_language_id
FROM Movie m
LEFT JOIN Genre g ON m.ID = g.ID
LEFT JOIN Spoken_language s ON m.ID = s.ID;
```

The problem with this query lies in its attempt to retrieve associated genre names and spoken languages. It uses LEFT JOIN operations to connect the Movie table with the Genre and Spoken_language tables based on their respective IDs. However, this approach can lead to data duplication and incorrect results, especially if a movie has multiple genres or spoken languages.

Unproblematic Query:

```
SELECT
  (SELECT STRING_AGG(g.Genre_id::TEXT, ',') FROM Genre g WHERE g.ID = m.ID) AS Genres,
  (SELECT STRING_AGG(s.Spoken_language_id::TEXT, ',') FROM Spoken_language s WHERE s.ID = m.ID) AS Spoken_Languages
FROM Movie m;
```

This optimized query resolves this issue by using subqueries to concatenate the genre names and spoken languages directly associated with each movie. Instead of joining tables and potentially duplicating data, it individually retrieves and concatenates the genre names and spoken languages for each movie using the STRING_AGG function. This ensures that each movie's genres and spoken languages are accurately represented in a concatenated format, providing a more reliable and concise result.

→ *Query 3:*

Let's say we want to retrieve the average rating (Vote_avg) for each genre.

Query | Query History

```
--problemetic query
SELECT
  gn.Genre_name,
  AVG(i.Vote_avg) AS Average_Rating
FROM Genre g
LEFT JOIN Movie m ON g.ID = m.ID
LEFT JOIN Imdb i ON m.Imdb_id = i.Imdb_Id
LEFT JOIN genre_name gn ON g.genre_id = gn.genre_id
GROUP BY gn.Genre_name;
```

The problem with this query lies in its calculation of the average rating for each genre. It uses LEFT JOIN operations to connect the Genre table with the Movie and IMDB tables based on their respective IDs. However, because LEFT JOIN includes all rows from the left table (Genre) and matching rows from the right tables (Movie and IMDb), it may include NULL values for movies that do not have corresponding ratings in the IMDB table. This can skew the average rating calculation.

Unproblematic Query:

```
--unproblamatic query
SELECT
  gn.Genre_name,
  AVG(i.Vote_avg) AS Average_Rating
FROM Genre g
LEFT JOIN Movie m ON g.ID = m.ID
LEFT JOIN Imdb i ON m.Imdb_id = i.Imdb_Id
LEFT JOIN genre_name gn ON g.genre_id = gn.genre_id
WHERE i.Vote_avg IS NOT NULL
GROUP BY gn.Genre_name;
```

This optimized query resolves this issue by adding a WHERE clause to filter out rows where the IMDB vote average (Vote_avg) is NULL. This effectively removes movies without ratings from the calculation of the average rating for each genre. By excluding these NULL values before calculating the average, the second query ensures that only movies with valid ratings contribute to the genre's average rating calculation, resulting in a more accurate representation of the average rating for each genre.

VII. REFERENCES

1. Database-System-Concepts-7th-Edition
2. Ramakrishnan - Database Management Systems 3rd Edition
3. Lecture notes