Unleashing Movie Insights

Converting MovieLens dataset into a SQL Database

Rohan Patel
ESDS
University At Buffalo
Buffalo, NY
rpatel38@buffalo.edu

Jay Thanki
ESDS
University At Buffalo
Buffalo, NY
jayyoges@buffalo.edu

***Abstract*— The MovieLens dataset is a valuable source of movie-related information, including 20 million movie ratings, 465,000 tag applications, and substantial user and movie data. However, it needs the hierarchical organization required for successful querying and analysis in its raw form. The issue is converting this dataset into an SQL database, allowing easy data administration, exploration, and analytics. This project aims to convert the MovieLens dataset into a well-structured SQL database to improve accessibility, analysis, and value to the film industry.**

Keywords— SQL, ER Model, Relational Model, MovieLens Data, Data Administration

I. Background

The MovieLens dataset contains much information on user preferences, movie genres, etc. However, it is a complicated collection of data that requires an organized format to be fully utilized. The inability to run meaningful queries, infer user preferences, and assess movie trends is hampered by the need for a well-organized database. We can simplify thorough data exploration and analysis by transforming this dataset into an SQL database, enabling data-driven decision-making in the film business.

A. Potential of the Project to Contribute to the Problem Domain

- **Data Accessibility**: A SQL database structure facilitates data access and querying for movie industry stakeholders, academics, and analysts. This increased accessibility is critical for data-driven decision-making.
- **Analytics and Insights**: The database supports a variety of queries, such as finding top-rated movies, determining user preferences, uncovering patterns across movie genres, and doing demographic analytics. This data helps to inform better content production, marketing tactics, and distribution decisions.
- **Business Strategy**: The initiative helps the film business to make data-driven strategic decisions. Executives may better utilize the SQL database insights to align production and marketing plans with customer preferences, increasing profitability, competitiveness, and efficiency.

B. Target Users

- **Data Analysts and Researchers**: The database will be used by analysts and researchers in the film business and academia for in-depth study on market trends, consumer preferences, and genre popularity.
- **Business Executives**: Executives in the film business will rely on the database's insights to make strategic choices affecting content production, marketing, and distribution.
- **Database Administrators**: Database administrators will oversee and maintain the SQL database, ensuring data correctness and timely updates.

C. Real World Scenario

In a real-world situation, a data analyst working for a movie production firm can perform market research using the SQL database created from the MovieLens dataset. They conduct queries to detect new movie genres, learn how user tastes differ by demographics, and examine historical trends in movie ratings. These findings influence the company's choice to invest in specific genres, generate targeted marketing efforts, and create content based on user preferences. Database administrators are responsible for ensuring the integrity of the database and keeping it up to date for these analyses.

II. E-R Diagram

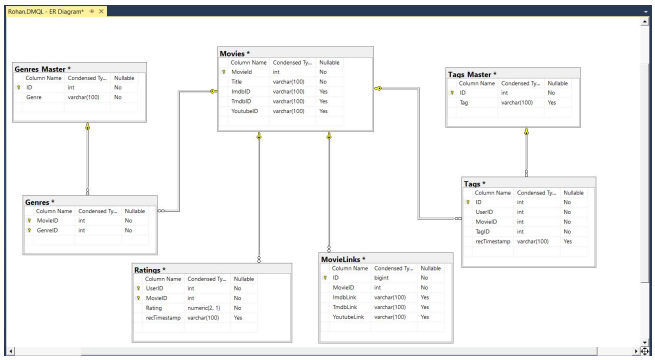


Fig. 1. ER diagram of the database

III. LIST OF RELATIONS AND THEIR ATTRIBUTES

A. Movies

- **MovieID(INT NOT NULL):** MovieID is an identifier for movies used by <https://movielens.org>. E.g., the movie Toy Story has the link <https://movielens.org/movies/1>.
- **Title(VARCHAR(1000) NOT NULL):** Title of the Movie and includes the year of release in parentheses.
- **ImdbID(VARCHAR(100) NULL):** ImdbId is an identifier for movies used by <http://www.imdb.com>. E.g., the movie Toy Story has the link <http://www.imdb.com/title/tt0114709/> for the imdbID: 0114709
- **TmdbID(VARCHAR(100) NULL):** TmdbId is an identifier for movies used by <https://www.themoviedb.org>. E.g., the movie Toy Story has the link <https://www.themoviedb.org/movie/862> for the tmdbID: 862
- **YoutubeID(VARCHAR(100) NULL):** YoutubeID is an identifier for the movies trailer on youtube by <https://www.youtube.com/watch?v=ID>. E.g., the movie Toy Story has the link <https://www.youtube.com/watch?v=K26_sDKnvMU> for the YoutubeID: K26_sDKnvMU
- **PK: MovieID** (Movie is able to uniquely identify every movie)

B. Genres Master

- **ID(INT NOT NULL):** Id stores the Id of each unique Genre
- **Genre(VARCHAR(100) NOT NULL):** Genre stores each unique genre for the movies
- **PK: ID** (ID uniquely identifies each genre. Primary key was not created on Genre because it is a varchar column and query runs very slow for the text data compared to the numeric data types for the joins).

C. Genres

- **MovieID(INT NOT NULL):** MovieID of the Movie referenced from Movies table
- **GenreID(INT NOT NULL):** GenreID of the Movie referenced from Genres_Master table
- **PK: MovieID, GenreID** (Combination of MovieID and GenreID is unique and so did not add ID column as primary key because it would be of not much use)
- **FK: MovieID** is referenced from the Table: Movies, Column: MovieID. Action: Cascade Delete

D. Tags Master

- **ID(INT NOT NULL)**: Id stores the id of each unique Tag.
- **Tag(VARCHAR(100) NOT NULL):** Tag stores all tags applied across all the movies
- **PK: ID** (ID is added for each tag. Primary key was not created on Tag itself as it is a varchar column as joins would run slow on the table for a varchar column compared to the int column)

E. Ratings

- **UserID(INT NOT NULL):** MovieLens users with their ids, which have been anonymized.
- **MovieID (INT NOT NULL):** MovieID of the Movie referenced from Movies table.
- **Rating(NUMERIC(2,1) NOT NULL):** Ratings given by a user for the movie out of 5.
- **recTimeStamp(VARCHAR(100) NULL):** Record Timestamp stores the Timestamps. It stores seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. Explicit timestamp insert is not allowed so instead of Timestamp datatype, varchar(100) is considered.
- **PK: UserID, MovieID** (Combination of UserId and MovieID uniquely identifies every row as each user has given only single rating for a movie)
- **FK: MovieID** is referenced from the Table: Movies, Column: MovieID. Action: Cascade Delete

F. Tags

- **ID(INT NOT NULL):** ID is primary key of the table to uniquely identify each tag applied by the user to the movie that user reviewed.
- **UserID(INT NOT NULL):** MovieLens users with their ids, which have been anonymized.
- **MovieID(INT NOT NULL):** MovieID of the Movie referenced from Movies table.
- **TagID(INT NOT NULL):** TagID stores the Id of the Tag referenced from Tags_Master table
- **recTimeStamp(VARCHAR(100) NULL):** Record Timestamp stores the Timestamps. It stores seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. Explicit timestamp insert is not

allowed so instead of Timestamp datatype, varchar(100) is considered.

- **PK: ID** (ID column is added as the primary key as it was not possible to uniquely identify a row with userid, movieID as single user have attached multiple tags to the same movie)

- **FK: MovieID** is referenced from the Table: Movies, Column: MovieID. Action: Cascade Delete. **TagID** is referenced from the Table: Tag_Master, Column: ID. Action: Cascade Delete

G. *MovieLinks*

- **ID(INT NOT NULL):** ID is primary key of the table to uniquely identify every movie

- **MovieID(INT NOT NULL):** MovieID of the Movie referenced from Movies table.

- **ImdbLink(VARCHAR(100) NULL)**: Full ImdbLink of the Movie

- **TmdbLink(VARCHAR(100) NULL)**: Full TmdbLink of the Movie

- **YoutubeLink(VARCHAR(100) NULL)**: Full Youtube Trailer Link of the Movie

- **PK: ID** (ID is added to identify a movie instead of MovieID as there were a lot of gaps in the MovieID which would create gaps in the index and make it run a bit slower)

- **FK: MovieID** is referenced from the Table: Movies, Column: MovieID. Action: Cascade Delete

IV. POPULATED DATABASE

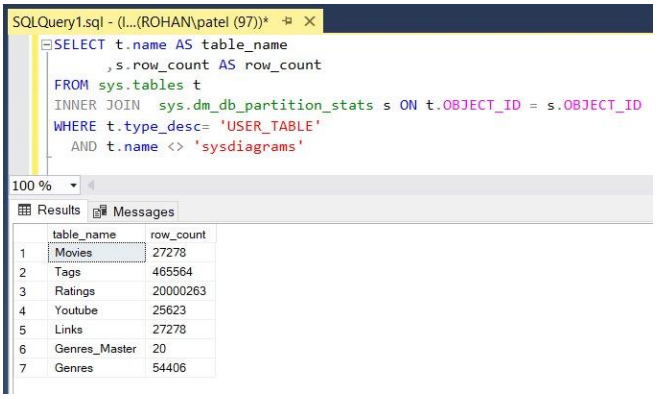


Fig. 2. Populated database

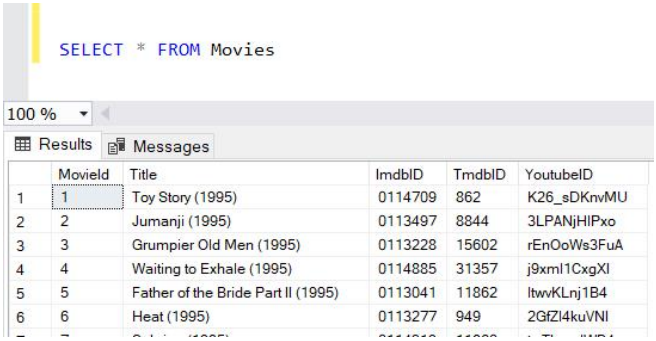


Fig. 3. Movies Table

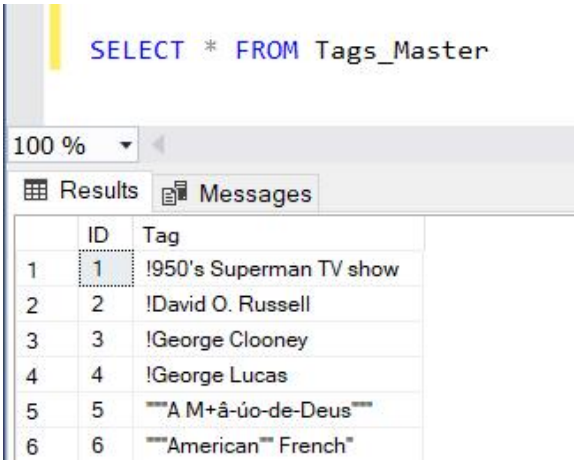
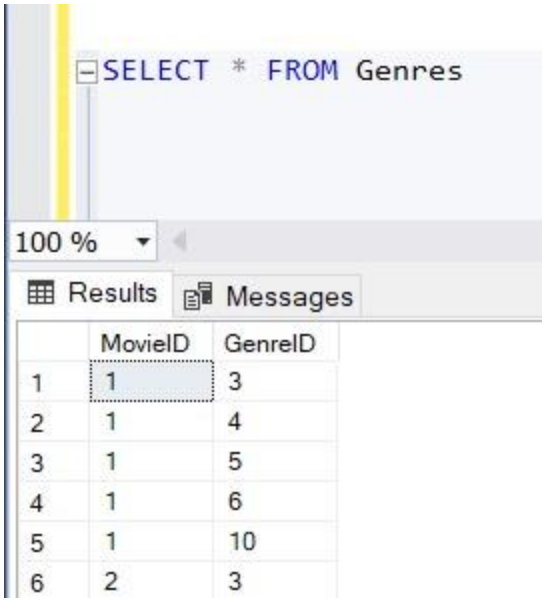


Fig. 4. Tags Master Table

Fig. 5. Genres Master Table

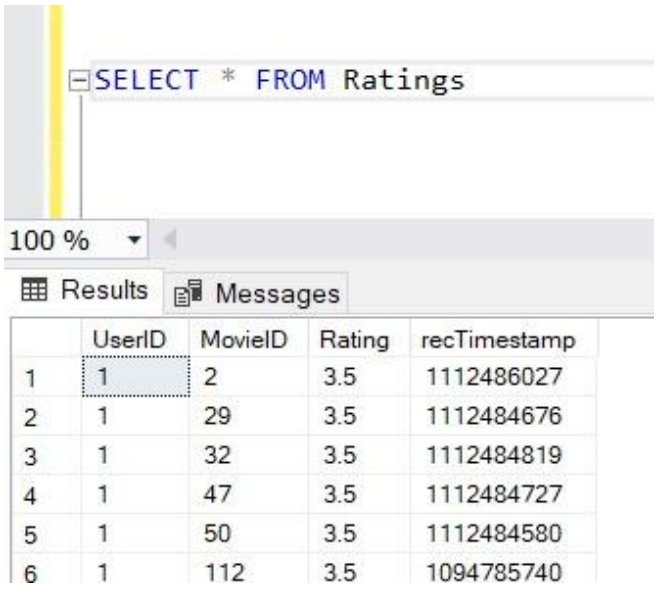
	TagID	UserID	MovieID	Tag	recTimestamp
1	1	18	4141	Mark Waters	1240597180
2	2	65	208	dark hero	1368150078
3	3	65	353	dark hero	1368150079
4	4	65	521	noir thriller	1368149983
5	5	65	592	dark hero	1368150078
6	6	65	668	bollywood	1368149876
7	7	65	898	screwball comedy	1368150160

Fig. 6. Tags Table



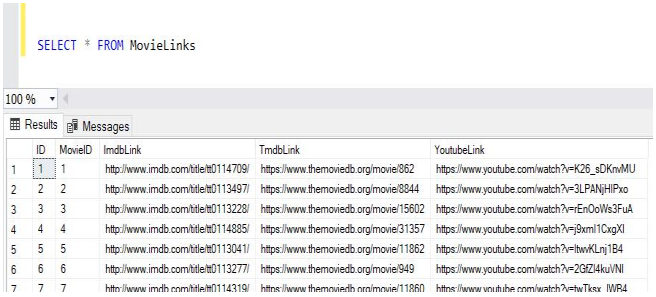
	MovieID	GenreID
1	1	3
2	1	4
3	1	5
4	1	6
5	1	10
6	2	3

Fig. 7. Genres Table



	UserID	MovieID	Rating	recTimestamp
1	1	2	3.5	1112486027
2	1	29	3.5	1112484676
3	1	32	3.5	1112484819
4	1	47	3.5	1112484727
5	1	50	3.5	1112484580
6	1	112	3.5	1094785740

Fig. 8. Ratings Table



	ID	MovieID	ImdbLink	TmdbLink	YoutubeLink
1	1	1	http://www.imdb.com/title/tt0114709/	https://www.themoviedb.org/movie/862	https://www.youtube.com/watch?v=K26_sDKnvMU
2	2	2	http://www.imdb.com/title/tt0113497/	https://www.themoviedb.org/movie/8844	https://www.youtube.com/watch?v=3LPANjHlPxo
3	3	3	http://www.imdb.com/title/tt0113228/	https://www.themoviedb.org/movie/15602	https://www.youtube.com/watch?v=rEnOoWs3FuA
4	4	4	http://www.imdb.com/title/tt0114885/	https://www.themoviedb.org/movie/31357	https://www.youtube.com/watch?v=j9xml1CxgXI
5	5	5	http://www.imdb.com/title/tt0113041/	https://www.themoviedb.org/movie/11862	https://www.youtube.com/watch?v=ltwvKLnj1B4
6	6	6	http://www.imdb.com/title/tt0113277/	https://www.themoviedb.org/movie/949	https://www.youtube.com/watch?v=2GfZI4kuVNI
7	7	7	http://www.imdb.com/title/tt0114319/	https://www.themoviedb.org/movie/11860	https://www.youtube.com/watch?v=twTkxx_IWR4

Fig. 9. Movie Links Table

V. PROOF OF BCNF COMPLIANCE

To prove that a given functional dependency (FD) is in Boyce-Codd Normal Form (BCNF), we need to ensure that it satisfies the following conditions:

1. The FD is non-trivial (i.e., the right-hand side is not a subset of the left-hand side).
2. The left-hand side of the FD is a superkey in the relation.

A. Movies (MovieId, Title, ImdbID, TmdbID, YoutubeID)

FD: {MovieId -> Title, ImdbID, TmdbID, YoutubeID}

1. Triviality: The FD {MovieId -> Title, ImdbID, TmdbID, YoutubeID} is non-trivial because the right-hand side (Title, ImdbID, TmdbID, YoutubeID) is not a subset of "MovieId."

2. Superkey: Since "MovieId" is the primary key of the "Movies" relation, it is a superkey because it is already a key attribute.

Since the given FD {MovieId -> Title, ImdbID, TmdbID, YoutubeID} is non-trivial, and "MovieId" is a superkey, it satisfies the conditions for BCNF. Therefore, we can conclude that the given FD is in BCNF.

B. Genres (MovieID, GenreID)

FD: {MovieID -> GenreID, GenreID -> MovieID}

Now, we need to check if this FD satisfies the abovementioned conditions.

1. Triviality:

- FD1: {MovieID -> GenreID} is non-trivial because "GenreID" is not a subset of "MovieID."
- FD2: {GenreID -> MovieID} is also non-trivial because "MovieID" is not a subset of "GenreID."
- Both FDs are non-trivial.

2. Superkey:

- MovieID is a superkey because it uniquely determines GenreID (FD1: MovieID → GenreID).
- GenreID is also a superkey because it uniquely determines MovieID (FD2: GenreID → MovieID).

Since the FDs are not trivial, but both the left-hand side attributes (MovieID and GenreID) are superkeys, the Genres (MovieID, GenreID) relation satisfies the conditions for BCNF.

C. Genres Master (ID, Genre)

FD: {ID -> Genre}

Now, we need to check if this FD satisfies the abovementioned conditions.

1. Triviality:

In this case, the FD is trivial because the right-hand side (Genre) is entirely contained within the left-hand side (ID).

2. Superkey:

The ID attribute is the primary key, uniquely identifying each tuple in the relation.

Since the FD is trivial, and the left-hand side attribute (ID) is a superkey, the Genres Master (ID, Genre) relation satisfies the conditions for BCNF. This indicates the relation is well-structured and free from redundancy or anomalies related to functional dependencies.

D. Tags (ID, UserID, MovieID, TagID, recTimestamp)

FD: {ID -> UserID, MovieID, TagID, recTimestamp}

Now, we need to check if this FD satisfies the abovementioned conditions.

1. Triviality:

In this case, the FD is non-trivial because the right-hand side (UserID, MovieID, TagID, recTimestamp) is not a subset of the left-hand side (ID).

2. Superkey:

Since ID is the primary key, it uniquely identifies each tuple in the relation. Therefore, ID is a superkey.

Since the FD is non-trivial, and the left-hand side attribute (ID) is a superkey, the Tags (ID, UserID, MovieID, TagID, recTimestamp) relation satisfies the conditions for BCNF. This indicates the relation is well-structured and free from redundancy or anomalies related to functional dependencies.

E. MovieLinks (ID, MovieID, ImdbID, TmdbID, YoutubeID)

FD: {ID -> MovieID, ImdbID, TmdbID, YoutubeID, MovieID -> ImdbID, TmdbID, YoutubeID,

ImdbID -> MovieID, TmdbID, YoutubeID}

Now, we need to check if this FD satisfies the abovementioned conditions.

1. Triviality:

The FDs are not trivial because the right-hand side of each FD contains attributes that are not subsets of the left-hand side. For example, in FD1, ID uniquely determines MovieID, ImdbID, TmdbID and YoutubeID, which are not a subset of ID.

2. Superkey:

The left-hand side of each FD is a superkey in the relation:

- ID is the primary key, so it uniquely identifies each tuple in the relation.
- ImdbID, TmdbID and YoutubeID and MovieID, TmdbID and YoutubeID are uniquely determined by the attributes MovieID, and ImdbID. Therefore, each of these attributes is also a superkey.

Since the FDs are not trivial, and the left-hand side attributes (ID, MovieID, and ImdbID) are superkeys, the MovieLinks (ID, MovieID, ImdbID, TmdbID, YoutubeID) relation satisfies the conditions for BCNF. This indicates the relation is well-structured and free from redundancy or anomalies related to functional dependencies.

F. Tags_Master(ID, Tag)

FD: {ID -> Tag}

Now, we need to check if this FD satisfies the abovementioned conditions.

1. Triviality:

In this case, the FD is trivial because the right-hand side (Tag) is entirely contained within the left-hand side (ID).

2. Superkey:

The ID attribute is the primary key, uniquely identifying each tuple in the relation.

Since the FD is trivial, and the left-hand side attribute (ID) is a superkey, the Tags Master(ID, Tag) relation satisfies the conditions for BCNF. This indicates the relation is well- structured and free from redundancy or anomalies related to functional dependencies.

G. Ratings (UserID, MovieID, Rating, recTimestamp)

FD: {UserID, MovieID -> Rating, recTimestamp}

Now, we need to check if this FD satisfies the above mentioned conditions.

1. Triviality:

The FD is non-trivial because the right-hand side ({Rating, recTimestamp}) is not a subset of the left-hand side ({UserID, MovieID}).

2. Superkey:

- The left-hand side of the FD ({UserID, MovieID}) is a superkey in the relation. Since UserID and MovieID form the primary key, they can uniquely identify each tuple in the relation.
- Since the FD is non-trivial, and the left-hand side ({UserID, MovieID}) is a superkey
- Since the FD is non-trivial and the left-hand side ({UserID, MovieID}) is a superkey, the Ratings(UserID, MovieID, and Rating, recTimestamp) relation satisfies the conditions for BCNF. This indicates the relation is well-structured and free from redundancy or anomalies related to functional dependencies.

REFERENCES

[1] https://grouplens.org/datasets/movielens/20m/

[2] https://grouplens.org/datasets/movielens/20m-youtube/