University of Tartu

Faculty of Science and Technology

Institute of Computer Science

Department of Information Technology

Movie Data Analysis

Project Part 1

Max Sebastian Segerkrantz, Kevin Christian Eriksson, Oto Pruul, Siim Turban

Tartu 2025

1. Business Brief

Objective:

The aim of the project is to use large-scale movie data (from 2000 until today) to answer business questions about movie performance, trends and impact.

Stakeholders:

- Product & Analytics teams at a streaming service
- Business Intelligence / Finance teams
- Content acquisition and programming teams
- Data Science / Recommendation system team

Key Performance Metrics:

- Average rating (weighted) by movie, genre, and time period
- Box office / revenue trends (total and per-movie) by release year and region
- Audience Activity (votes & rating Changes)

Business Questions:

- Which genres have the highest average ratings?
- Which directors consistently produce high-rated movies and high revenue?
- How does average rating correlate with box-office revenue across release years?
- What are the top 10 movies by revenue per genre for a given year?
- How does the runtime affect changes to the overall box-office revenue?

2. Datasets

Primary datasets used (public):

1. **The TMDb** (The Movie Database) is a comprehensive movie database that provides information about movies, including details like titles, ratings, release dates, revenue, genres, and much more.

   https://www.kaggle.com/datasets/asaniczka/tmdb-movies-dataset-2023-930k-movies

2. **IMDb non-commercial datasets** (IMDb data files in TSV: title.ratings, title.crew,) – authoritative movie identifiers, ratings (averageRating, numVotes), crew and name metadata.

   https://developer.imdb.com/non-commercial-datasets/

3. Tooling

Ingestion / Loading:

- Docker to containerize the environment and make dataset ingestion repeatable.
- Download raw IMDb .tsv files and Kaggle CSVs manually (or with simple scripts) into a local Docker volume.
- Version control and collaboration managed through GitHub (project repository, scripts, schema files, and documentation). https://github.com/kevineriksson/Project-1

Storage / Database:

- Use Postgres (running inside Docker) as the main database to store both the staging data and the final star schema.
- Tables in Postgres will represent both the staging layer (raw tables from IMDb and Kaggle) and the warehouse layer (star schema with fact and dimension tables).

Data Modeling:

- ER diagrams to design the logical structure of how datasets relate.
- Translate the ER design into a dimensional star schema inside Postgres.

Analytics / Querying:

- Run SQL queries directly on Postgres (pgAdmin).
- Answer the business questions using the star schema tables.

4. Data Architecture

Data flow:

IMDb TSVs + Kaggle CSV → Ingestion (manual download / Docker volume) → Postgres staging tables → Star schema (Postgres) → SQL queries & reporting

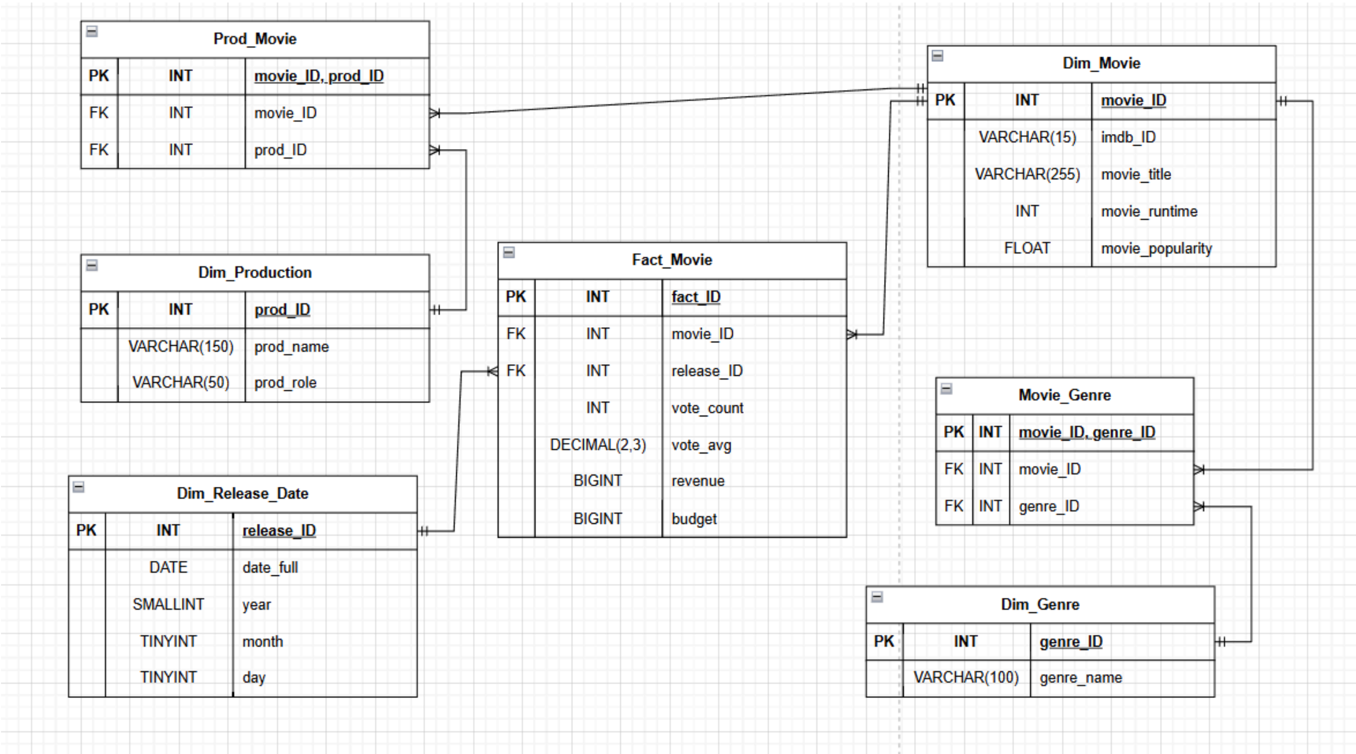
Ingestion methods & frequency:

- IMDb datasets: Manual download from datasets.imdbws.com, then load TSV into Postgres staging tables.
- Kaggle dataset: CSV export and load into Postgres staging tables.
- Frequency: For this project, data is refreshed daily (e.g., each time the Docker container is started). In a production setup, this would be automated through scheduled daily loads.

Data quality checks:

- Null check: fact_ID must not be NULL in fact table.
- Uniqueness check: imdb_ID must be unique in the movie dimension table.
- Vote average check: ratings between 0 and 10.

5. Data model



- **High granularity**: One row per unique combination of movie + release date.
- Dim_release_date: **STATIC** because the release date is fixed and unchanging.
- Dim_genre: **STATIC** because genres usually do not change after classification.
- Dim_production: **TYPE 2** because we might want to keep track of a production company's previous names to associate or compare performance before or after the name change.
- Dim_movie: **TYPE 1** because, for now, we don't need to keep track of title- or slight runtime changes. Also, popularity data changes very often and in this case is not needed for historical analysis.

6. Data dictionary

Fact table: Fact_Movie

Column	Description	Data Type
fact_ID	Unique identifier for fact table row	INT
movie_ID	Foreign key to Dim_Movie	INT
release_ID	Foreign key to Dim_Release_Date	INT
vote_count	Total number of votes on TMDB	INT
vote_avg	Average rating (0-10 scale)	DECIMAL(2,3)
budget	Budget of the movie (from TMDB)	BIGINT
revenue	Revenue generated by the movie (from TMDB)	BIGINT

Junction table: Prod_movie

Column	Description	Data Type
movie_ID, prod_ID	Composite primary key	INT
prod_ID	Foreign key to Dim_Movie table	INT
movie_ID	Foreign key to Dim_Production table	INT

Dimension table: Dim_Production

Column	Description	Data Type
prod_ID	Unique identifier for the producer entity	INT
production_name	Name of the production company or person	VARCHAR(150)
production_role	Role (e.g., director, producer, writer)	VARCHAR(50)

Dimension table: Dim_Release_date

Column	Description	Data Type
release_ID	Unique identifier for the release date	INT
date_full	Full date of release	DATE
year	Year of the release	SMALLINT
month	Month of the release	TINYINT
day	Day of the release	TINYINT

Dimension table: Dim_Genre

Column	Description	Data Type
genre_ID	Unique identifier for the genre	INT
genre_name	Name of the genre	VARCHAR(100)

Junction table: Movie_Genre

Column	Description	Data Type
movie_ID, genre_ID	Composite key primary key	INT
movie_ID	Foreign key to Dim_Movie table	INT
genre_ID	Foreign key to Dim_Genre table	INT

Dimension table: Dim_Movie

Column	Description	Data Type
movie_ID	Unique identifier for the movie	INT
imdb_ID	IMBd identifier for the movie	VARCHAR(15)
movie_title	Title of the movie	VARCHAR(255)
movie_runtime	Duration of the movie (in minutes)	INT
movie_popularity	Popularity score given from TMBD	FLOAT

7. Demo queries

- **Which genres have the highest average ratings?**

  SELECT g.genre_name, ROUND(AVG(f.vote_avg), 2) AS avg_rating FROM Fact_Movie f JOIN Movie_Genre mg ON f.movie_ID = mg.movie_ID JOIN Dim_Genre g ON mg.genre_ID = g.genre_ID GROUP BY g.genre_name ORDER BY avg_rating DESC;

- **Which directors consistently produce high-rated movies and high revenue?**

  SELECT p.prod_name AS director, COUNT(DISTINCT f.movie_ID) AS total_movies, ROUND(AVG(f.vote_avg), 2) AS avg_rating, ROUND(AVG(f.revenue), 0) AS avg_revenue FROM Fact_Movie f JOIN Prod_Movie pm ON f.movie_ID = pm.movie_ID JOIN Dim_Production p ON pm.prod_ID = p.prod_ID WHERE p.prod_role = 'Director' GROUP BY p.prod_name HAVING COUNT(DISTINCT f.movie_ID) >= 3 ORDER BY avg_rating DESC, avg_revenue DESC;

- **How does average rating correlate with box-office revenue across release years?**

  SELECT r.year, ROUND(AVG(f.vote_avg), 2) AS avg_rating, ROUND(AVG(f.revenue), 0) AS avg_revenue FROM Fact_Movie f JOIN Dim_Release_Date r ON f.release_ID = r.release_ID GROUP BY r.year ORDER BY r.year;

- **What are the top 10 movies by revenue per genre for a given year?**

  SELECT g.genre_name, m.movie_title, f.revenue FROM Fact_Movie f JOIN Dim_Movie m ON f.movie_ID = m.movie_ID JOIN Movie_Genre mg ON f.movie_ID = mg.movie_ID JOIN Dim_Genre g ON mg.genre_ID = g.genre_ID JOIN Dim_Release_Date r ON f.release_ID = r.release_ID WHERE r.year = 2023 QUALIFY ROW_NUMBER() OVER (PARTITION BY g.genre_name ORDER BY f.revenue DESC) <= 10;

- **How does the runtime affect changes to the overall box-office revenue?**

  SELECT ROUND(m.movie_runtime / 10) * 10 AS runtime_bucket, -- group by 10-minute bins ROUND(AVG(f.revenue), 0) AS avg_revenue, COUNT(*) AS movie_count FROM Fact_Movie f JOIN Dim_Movie m ON f.movie_ID = m.movie_ID

  GROUP BY runtime_bucket ORDER BY runtime_bucket;

8. Roles and contribution

All active team members contributed equally (approximately 33% each). Siim was unable to participate in this phase due to a military exercise call-up and will contribute more in later project parts. All participating members collaborated on dataset identification and defining the overall project direction.

- **Max** – Team coordination, business brief, demo SQL queries, final document review, and corrections.
- **Kevin** – Tooling & Github setup, data architecture, and data dictionary.
- **Oto** – Data model design.

9. LLM Disclosure

- **Many-to-many relationship question**
  https://chatgpt.com/share/68e03837-022c-8009-87d2-238c2b452d70

- **Business question**
  https://chatgpt.com/share/68e272a5-062c-8009-b374-9136527dd055

- **Checking compatibility between the schema and business goals (member has no GPT account, so screenshots instead)**

✅ 1. Supports All Business Goals?

Let's quickly map your schema to your original questions:

Business Goal	Supported?	How
Highest-rated genres	✅	`Fact_Movie` → `Movie_Genre` → `Dim_Genre`, group by `vote_avg`
Top directors by rating and revenue	✅	`Prod_Movie` (filter `prod_role = 'director'`) + `Fact_Movie`
Rating vs. revenue over time	✅	`Fact_Movie` + `Dim_Release_Date`
Top 10 movies by revenue per genre for a given year	✅	Join `Fact_Movie`, `Movie_Genre`, `Dim_Genre`, `Dim_Release_Date`
Rating distributions by region	❌ Skipped	You've decided to ignore this — acceptable
Runtime vs. box-office	✅	`Dim_Movie.movie_runtime` + `Fact_Movie.revenue`

Thanks for the update! Your schema is now **very well-designed**, and it aligns effectively with your defined business goals. You've:

- ✅ Removed redundant `genre_ID` from `Fact_Movie`
- ✅ Normalized many-to-many relationships (`Movie_Genre`, `Prod_Movie`)
- ✅ Clarified granularity of facts (movie + release date)
- ✅ Clearly separated dimensions and facts
- ✅ Defined composite primary keys for junction tables

That's a strong schema for analytical workloads and star-schema-based warehousing.

- **Checking if the schema design is proper.**

✅ 2. Schema Design Quality

Checkpoint	Status	Comment
Clear fact-dimension separation	✅	No metrics in dimensions ✔
Normalized multi-value fields	✅	Genres and production roles ✔
Composite PKs in junctions	✅	Defined in `Movie_Genre`, `Prod_Movie` ✔
Proper data types (implied)	⚠	Just make sure `vote_avg` is `FLOAT`, `revenue` / `budget` are `BIGINT`, etc.
Foreign keys exist for integrity	✅	All links are visible ✔
Date dimension is atomic	✅	Full date + breakdown ✔

- **Checking if crow's foot notation is valid.**

🗒 Summary Table

Relationship	Crow's Foot Correct?	Notes
`Dim_Movie` → `Fact_Movie`	✅ Yes	1:N, direct link
`Dim_Movie` → `Movie_Genre_junct`	✅ Yes	1:N, visual overlap with above
`Dim_Movie` → `Prod_Movie_junct`	✅ Yes	1:N
`Dim_Production` → `Prod_Movie_junct`	✅ Yes	1:N
`Dim_Genre` → `Movie_Genre_junct`	✅ Yes	1:N
`Dim_Release_Date` → `Fact_Movie`	✅ Yes	1:N

- **Checking if the Demo SQL queries are correct based on the star schema**
  https://chatgpt.com/share/68e241a8-ac6c-800f-abf7-7270403dc984

10. Github

Link to Github repository - https://github.com/kevineriksson/Project-1