Cloud Data Infrastructure
Chapter 2 : Denormalization

Juliette Danel
Polytechnic Institute of Paris
juliette.danel@ip-paris.fr

Godefroy Lambert
Polytechnic Institute of Paris
godefroy.lambert@ip-paris.fr

Marius Ortega
Polytechnic Institute of Paris
marius.ortega@ip-paris.fr

23 February 2024

1 Denormalized models

In the following section, we will discuss two denormalized model implementations of the ImDb database (cf. Chapter 1). After considering and analyzing each one of them, we will select the final denormalized schema for our project.

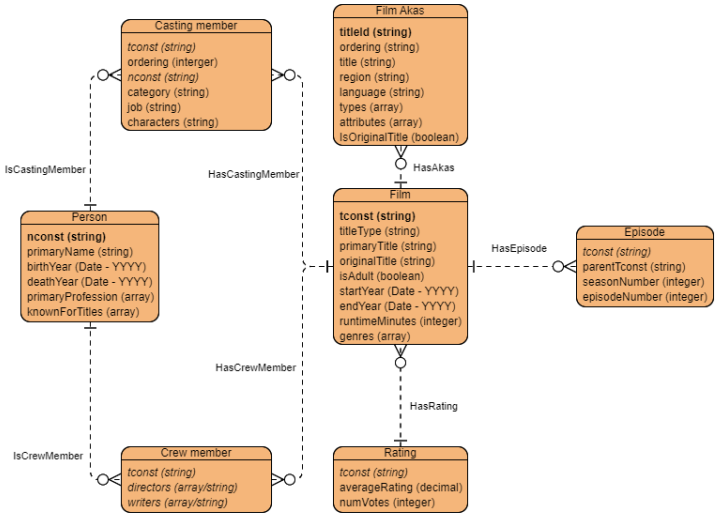

Figure 1: Entity-relationship diagram

We will base the denormalization on the original entity-relationship schema (figure 1).

1.1 Model 1

1.1.1 Choices & motivation

We noticed very strong interactions in our queries between the table Person and Film by the mean of either the table CrewMember or the table CastingMember which consists of a further motivation to tighten links between these tables. Thus, we overloaded Person with the most used attributes of CastingMember and CrewMember with respect to our queries, which are respectively primaryName and directors. The overload should increase performances of the database query responsiveness.

On another aspect, we chose to overload Akas table to create a cheaper access to region and isOriginalTitle attributes as they are often used in joins with Film table. We allow ourselves to add the attribute originalRegion.
We also create a faster access to all of a film's titles by adding them directly in the table Film.

Finally, we perform a fusion of Film and Rating (also applied in model 2, see section 1.2) as it mostly provides pros such as fast access to ratings which are often used. On the other side, it makes the film table heavier but still is an acceptable fusion as Rating table only relatively light attributes (int and decimal).

The denormalization steps for model 1 are summarized in the list bellow :

- Fusion : Film & Ratings : access rating without a join, as we frequently use this attribute
- Overload: Person.primaryName → CastingMember : to access faster to the person's name.
- Materialization : CastingMember.job → Person : one of the single attributes used w.r.t Person
- Overload : CrewMember.directors and CrewMember.writers → Film
- Materialization : Akas.region, Akas.title → Film : we put the original region directly in Film, we also put the list of all the titles.

1.1.2 Denormalized Json Schema

Not to make the document unreadable, we put the denormalization schemas on our GitHub page (see schema for model 1 here).

1.1.3 Denormalized EA diagram

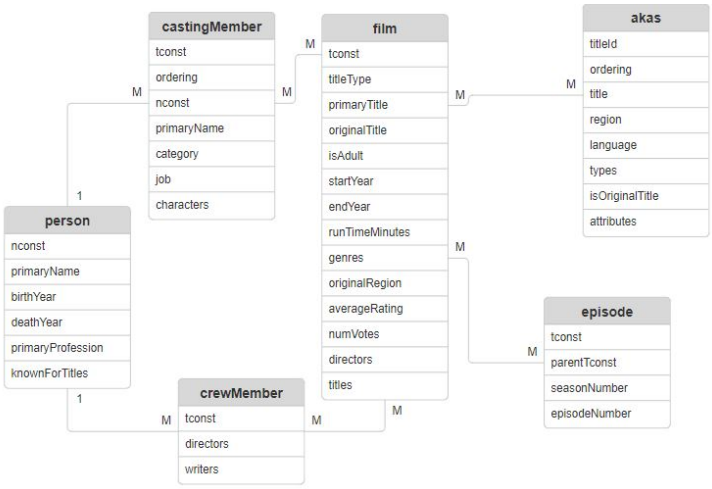

Figure 2: Entity-relationship diagram for model 1

1.2 Model 2

1.2.1 Choices & motivation

While studying the first denormalization (section 1.1), we noticed strong bounds between Person and Film tables which are still applicable. Consequently, this second model is an alternative way to treat these interactions. Here, we propose to overload CrewMember.directors as well as CastingMember.job in Film table. Therefore, the film will become much heavier as each film will have lists of job associated with nconst primary keys (same goes for attribute directors. To address that, we split the table Person to keep only light and much utilized attributes such as primaryName, birthYear, deathYear (used in 3 queries out of 8).

As explained for model 1 (section 1.1), the most interesting denormalization to perform on Rating table is, in our opinion, a Fusion with table Film in order to limit joins to that regard. Consequently, we also perform that denormalization operation for this model.

As a last step, we provide an alternative way to treat Akas table. Previously (section 1.1), we overloaded Film with attributes of interest from the Akas table. Now, we will split Akas to reduce the cost of the join between Akas and Film as is medium frequent operation we perform on the database (present in 2 queries out of 8).

Once again, we provide a summary of the denormalization operations we performed to create the second model :

- Fusion : Film & Ratings : access rating without a join, as we frequently use this attribute
- Overload : CrewMember.directors → Film
- Overload : CastingMember.job → Film
- Split Person : primaryName, birthYear, deathYear | others : keep only useful, light and frequently used attributes to facilitate the join of Person and Film.
- Split Akas : language, region, isOriginalTitle | others : As we rarely use it, no need to fusion it with Film. However, 3 of its attribute have a real importance for our use case. Thus, we want to access them in a cheaper way then what is possible with the normalized schema. In addition, Film is already relatively overloaded, so we won't make it any heavier.

1.2.2 Denormalized Json Schema

In the same way as for model 1, we provide the denormalized Json Schema at the following link.

1.2.3 Denormalized EA diagram

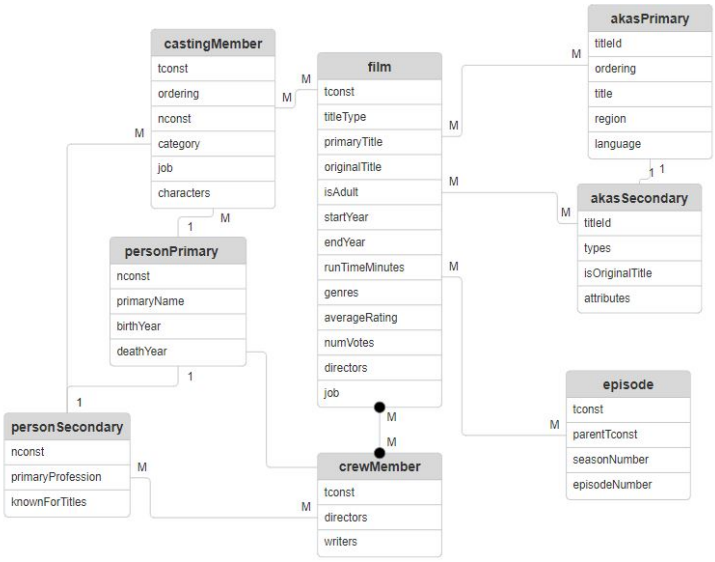

Figure 3: Entity-relationship diagram for model 1

2 Queries after denormalization

In this section, we take the update queries explanation from report 1 (link here), and adapt them to the new denormalized model. For each model, we only present queries that have been affected by the denormalization.

2.1 Queries for model 1

2.1.1 Select the top 10 movies of the year X shorter than 2 hours

Tables and attributes

film (tconst, startYear, primaryTitle, titleType, runTimeMinutes, averageRating, numVotes)

Projections

tconst, primaryTitle, startYear, ratingScore, titleType, runTimeminutes

Filters, and aggregations and operations

- titleType equals to "movie"
- startYear equals to the specific year we are looking for
- ratingScore equals to $averageRating * log(numVotes + 1)$
- runTimeMinutes is less than 120
- order by descending ratingScore
- keep the 10 first results

2.1.2 Actors that played in at least 3 adult films

Tables and attributes

- **film** (tconst, isAdult)
- **Person** (nconst, primaryName)
- **CastingMember** (nconst, tconst, job)

Joins

Join CastingMember with the rest on the pair tconst, nconst.

Projections

Person.nconst, Person.primaryName, count(Film.tconst)

Filters, aggregations and operations

- Filter Actors that played in at least 3 movies using table Person
- Left join Person and Film
- Unwind Person.job
- Group by **nconst**
- Count the number of **tconst**

2.1.3 Select the director that participated in the most title for each genre

Tables and attributes

- **film** (tconst, genres, directors)

Projections

tconst, genres, directors

Filters, aggregations and operations

- for each director, count the number of titles per genre
- for each genre, keep the director that has the most titles

2.1.4 Count the number of actors dead before the end of a series

Tables and attributes

- **CastingMember** (tconst, nconst, job)
- **Film** (tconst, startYear, endYear, typeTitle)

Joins

Inner join **CastingMember** and **Film** on job.tconst and tconst

Projections

Count(nconst)

Filters, aggregations and operations

- Filter on titleType = $tvSeries$
- Unwind Person.job so as to group by tconst
- Join Film and Person
- Group by tconst
- Having at least an actor for which deathYear >= startYear and deathYear <= endYear
- Count the output

2.1.5 Select the top Y favorite genre of the year X

Tables and attributes

Film (tconst, genre, startYear, averageRating, numVotes)

Joins

Left join Film and Rating on tconst

Projections

tconst, startYear, averageRating $\times \log(numVotes)$

Filters, aggregations and operations

- Filter film on year X
- Calculate the score : averageRating $\times \log(numVotes)$
- Order by score in descending order
- Keep the top Y results

2.1.6 Select the average number of movies released per decade per region

Tables and attributes

film (tconst, titleType, startYear, isOriginal, region)

Projections

tconst, titleType, startYear, isOriginal, region

Filters, aggregations and operations

- titleType equals to "movie"
- isOriginal equals to True
- modify the startYear to only keep the decade (ex: 1956 → 1950, 2020 → 2020, etc.)
- Unwind overloaded part of Akas in Film
- group by region and decade
- count the number of movies made by year for each decade

2.2 Queries for model 2

2.2.1 Select the top 10 movies of the year X shorter than 2 hours

Tables and attributes

film (tconst, startYear, primaryTitle, titleType, runTimeMinutes, averageRating, numVotes)

Projections

tconst, primaryTitle, startYear, ratingScore, titleType, runTimeminutes

Filters, and aggregations and operations

- titleType equals to "movie"
- startYear equals to the specific year we are looking for
- ratingScore equals to $averageRating * log(numVotes + 1)$
- runTimeMinutes is less than 120
- order by descending ratingScore
- keep the 10 first results

2.2.2 Select a title containing the string "XXX" in at least one of its titles

Tables and attributes

- **film** (tconst, primaryTitle, originalTitle)
- **filmAkas (primary)** (titleId, title)

Joins

Left join between **film** and **filmAkas**, left on tconst, right on titleId (here we join only on the split of interrest from the **filmAkas** table (i.e. the main split).

Projections

tconst, primaryTitle, originalTitle, title

Filters, aggregations and operations

- keep rows where primaryTitle or originalTitle or title contains the specific string we are looking for

2.2.3 Actors that played in at least 3 adult films

Tables and attributes

- **film** (tconst, isAdult, (nconst, job))
- **Person** (nconst, primaryName)

Joins

- Inner join Film and Person on **tconst**

Projections

Person.nconst, Person.primaryName, count(Film.tconst)

Filters, aggregations and operations

- Filter Film on isAdult = $True$
- Unwind Film.job
- Filter CrewMember on job = *Actor*
- Inner join Film and Person
- Group by **nconst**
- Count the number of **tconst**

2.2.4 Select the director that participated in the most title for each genre

Tables and attributes

- **film** (tconst, genres, directors)

Projections

tconst, genres, directors

Filters, aggregations and operations

- for each director, count the number of titles per genre
- for each genre, keep the director that has the most titles

2.2.5 Count the number of actors dead before the end of a series

Tables and attributes

- **Film** (tconst, startYear, endYear, typeTitle, (nconst, job))

Projections

Count(nconst)

Filters, aggregations and operations

- Filter on titleType = *tvSeries*
- Unwind Film.job
- Group by Film
- Having at least an actor for which deathYear >= startYear and deathYear <= endYear
- Count the output

2.2.6 Select the top Y favorite genre of the year X

Tables and attributes

Film (tconst, genre, startYear, averageRating, numVotes)

Projections

tconst, startYear, averageRating $\times \log(numVotes)$

Filters, aggregations and operations

- Filter film on year X
- Calculate the score : averageRating $\times \log(numVotes)$
- Order by score in descending order
- Keep the top Y results

2.2.7 Select the average number of movies released per decade per region

Tables and attributes

- **film** (tconst, titleType, startYear)
- **filmAkas (primary)** (titleId, isOriginal, region)

Joins

Inner join between **film** and **filmAkas**, left on tconst, right on titleId (here we only join on a subpart of Akas table, thanks to the splitting operation).

Projections

tconst, titleType, startYear, isOriginal, region

Filters, aggregations and operations

- titleType equals to "movie"
- isOriginal equals to True
- modify the startYear to only keep the decade (ex: 1956 $\rightarrow$ 1950, 2020 $\rightarrow$ 2020, etc.)
- group by region and decade
- count the number of movies made by year for each decade

3 Statistics after denormalization

Considering modifications from the E/A model 1 to the denormalized one, the most interesting aspect that have changed are cardinality of elements in each table. In case, you desire to have general statistics about the database, you can visit our first report's statistics section (link here).

3.1 Statistics for model 1

Given the denormalization steps (1.1.1), and since we are only doing fusion and overloading, the number of documents for all table remains unchanged. However, films documents and castingMember documents are bigger.

3.1.1 Film (updated)

We denote changes in cardinality of attributes for table Person :

- A film have an average of 1.19 directors.
- Each film have an average of 5.12 titles.

3.1.2 Casting Member (updated)

- Each person have an average of 9.5 characters.

3.2 Statistics for model 2

With this second data architecture, the table Film becomes even more central then previously as you can reach any information of the database with a single "join" operation from Film. As most of the other denormalization steps mostly involve splitting operations, we will focus this updated statistics section on the table Film. For any general information about the attributes cardinality, types and distribution, refer to our first report available on our github page.

First thing to notice are the changes of cardinality for the table Film :

- Each Film has an average of 6 persons in its workforce.
- But note that each Film only have 2 different job titles on average (the more precise number is 1.75).

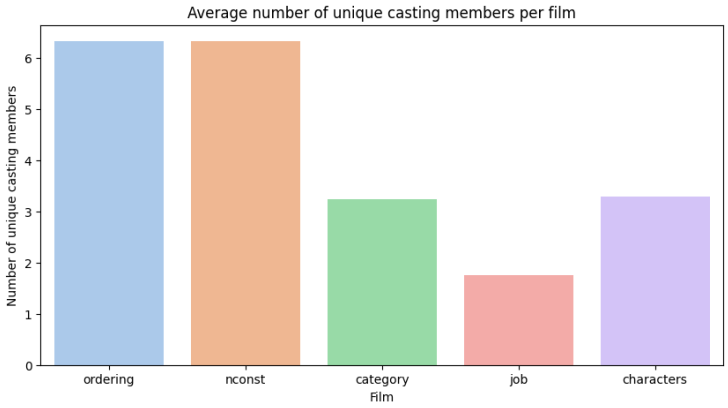

Figure 4: Barplot : Overloaded CrewMember attribute distribution in Film

4 Conclusion

To sum up, this report presents two denormalized models for the ImDb dataset.

The first one 1.1 allow us get access to directors and all title of the film without loading the documents. Adding actor name with their role in films also allow us to save some time on some requests. Change on this model are quite minor but allows us to speed a lot of requests.

The second model 1.2 is quiet different from the first one as it put the accent on the table Film instead of the table Person. This completely changes the dynamics of the data model and the way to query it. As exposed in section 3.2, this model allows to access any data from the Film table in a single join. The main drawback of the architecture is the consequent heaviness of the table Film. To mitigate that, we split Person and Akas to lighten the possible join.

As a final word, these two model share a major common point ; they remain relatively adaptable and are not too specific to our set of queries. In fact, it was important for us to aim beyond the short-term objectives of the project to create a durable model.