Cloud Data Infrastructure
Chapter 3 : Data Infrastructure Optimization

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

15 March 2024

1 Introduction

Considering the model and needs expressed in chapter 1 (link) and the denormalization performed in chapter 2 (link), we will now present how costly each denormalized model is. To do so, we will start by expressing our use cases in MQL (Mongo Query Language) and conclude on the efficiency of each model.

2 MQL queries

2.1 Model 1

In this section, we translate the previously selected queries into MongoDB Queries Language. We base the queries on the first model we produced in the last report (see Fig. 1 1).

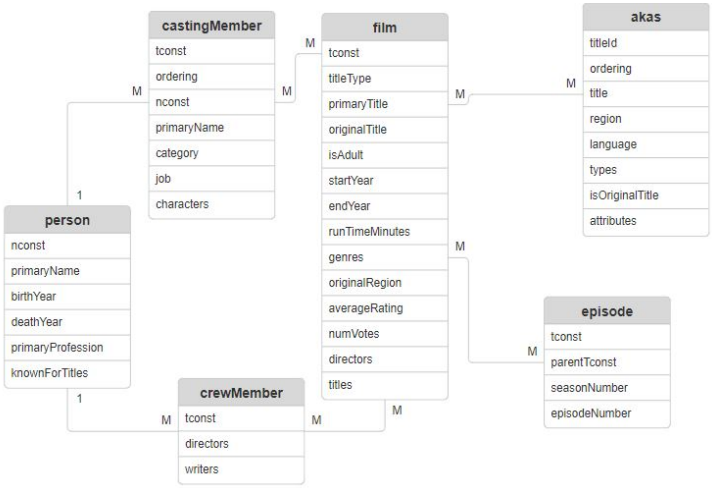

Figure 1: Entity-relationship diagram for model 1

2.1.1 Select the top 10 movies of the year X

To illustrate the query, we set the year to 1999.

```
db.film.aggregate([
  {
    $match: {
      "titleType": "movie",
      "startYear": 1999,
      "runTimeMinutes": { $lt: 120 }
    }
  },
  {
    $addFields: {
      "ratingScore": {
        $multiply: [
          "$averageRating",
          { $log: { $add: ["$numVotes", 1] } }
        ]
      }
    }
  },
  { $sort: { "ratingScore": -1 } },
  { $limit: 10 },
  {
    $project: {
```

```
            "_id": 0,
            "tconst": 1,
            "primaryTitle": 1,
            "startYear": 1,
            "ratingScore": 1,
            "runTimeMinutes": 1
          }
        }
      ])
```

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

To illustrate the query, we set the string to "love".

```
db.film.find(
  {
    $or: [
      { "primaryTitle": { $regex: "love", $options: "i" }
      ↪  },
      { "titles": { $regex: "love", $options: "i" } }
    ]
  },
  {
    "_id": 0,
    "tconst": 1,
    "primaryTitle": 1,
    "originalTitle": 1,
    "title": 1
  }
)
```

2.1.3 Actors that played in at least 3 adult films

```
db.CastingMember.aggregate([
  {
    $lookup: {
      from: "Person",
      localField: "nconst",
      foreignField: "nconst",
      as: "person"
    }
  },
```

```
    { $unwind: "$person" },
    {
      $lookup: {
        from: "Film",
        localField: "tconst",
        foreignField: "tconst",
        as: "film"
      }
    },
    { $unwind: "$person.job" },
    { $match: { "film.isAdult": true } },
    {
      $group: {
        "_id": "$person.nconst",
        "primaryName": { $first: "$person.primaryName" },
        "totalAdultFilms": { $sum: 1 }
      }
    },
    { $match: { totalAdultFilms: { $gte: 3 } } },
    {
      $project: {
        "_id": 0,
        "nconst": "$_id",
        "primaryName": 1,
        "totalAdultFilms": 1
      }
    }
  ])
```

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

```
db.film.aggregate([
  { $unwind: "$genres" },
  {
    $group: {
      "_id": { "genre": "$genres", "director":
      ↪  "$directors" },
      "titleCount": { $sum: 1 }
    }
  },
  { $sort: { "_id.genre": 1, "titleCount": -1 } },
  {
    $group: {
```

```
        "_id": "$_id.genre",
        "topDirector": { $first: "$_id.director" },
        "maxTitleCount": { $first: "$titleCount" }
      }
    },
    {
      $project: {
        "_id": 0,
        "genre": "$_id",
        "topDirector": 1,
        "maxTitleCount": 1
      }
    }
  ])
```

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

```
db.CastingMember.aggregate([
  {
    $lookup: {
      from: "Film",
      localField: "tconst",
      foreignField: "tconst",
      as: "film"
    }
  },
  { $unwind: "$film" },
  { $match: { "film.typeTitle": "tvSeries" } },
  { $unwind: "$job" },
  {
    $group: {
      "_id": "$film.tconst",
      "deadActorsCount": {
        $sum: {
          $cond: [
            { $and: [{ $eq: ["$job", "actor"] }, { $gte:
            ↪  ["$film.endYear", "$deathYear"] }, {
            ↪  $lte: ["$film.startYear", "$deathYear"]
            ↪  }] },
            1,
            0
          ]
        }
      }
```

```
        }
      },
      { $match: { "deadActorsCount": { $gt: 0 } } },
      { $count: "output" }
    ])
```

2.1.6 Select the top Y favorite genre of the year X

To illustrate the query, we set the top to 5 and the year to 2012.

```
db.Film.aggregate([
  {
    $lookup: {
      from: "Rating",
      localField: "tconst",
      foreignField: "tconst",
      as: "ratings"
    }
  },
  { $unwind: "$ratings" },
  { $match: { "startYear": 2012 } },
  {
    $addFields: {
      "score": { $multiply: ["$averageRating", { $ln: {
      ↪  $add: ["$ratings.numVotes", 1] } }] }
    }
  },
  { $sort: { "score": -1 } },
  { $limit: 5 },
  {
    $project: {
      "_id": 0,
      "tconst": 1,
      "startYear": 1,
      "score": 1
    }
  }
])
```

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

```
db.film.aggregate([
```

```
    {
      $match: {
        "titleType": "movie",
        "isOriginal": true
      }
    },
    {
      $addFields: {
        "decade": { $subtract: [ { $toInt: { $substr:
        ↪  ["$startYear", 0, 3] } }, { $mod: [ { $toInt: {
        ↪  $substr: ["$startYear", 0, 3] } }, 10 ] } ] }
      }
    },
    { $unwind: "$region" },
    {
      $group: {
        "_id": { "region": "$region", "decade": "$decade"
        ↪  },
        "movieCount": { $sum: 1 }
      }
    },
    {
      $group: {
        "_id": "$_id.region",
        "averageMoviesPerDecade": { $avg: "$movieCount" }
      }
    },
    {
      $project: {
        "_id": 0,
        "region": "$_id",
        "averageMoviesPerDecade": 1
      }
    }
  ])
```

2.1.8 Average number of appearance for an actor per series

```
db.castingMembers.aggregate([
  {
    $match: {
      "category": "actor"
    }
  },
```

```
  {
    $group: {
      "_id": {
        "actor": "$nconst",
        "series": "$tconst"
      },
      "appearances": { $sum: 1 }
    }
  },
  {
    $group: {
      "_id": "$_id.actor",
      "averageAppearances": { $avg: "$appearances" }
    }
  },
  {
    $sort: { "averageAppearances": -1 }
  },
  {
    $limit: 3
  },
  {
    $lookup: {
      from: "personPrimary",
      localField: "_id",
      foreignField: "nconst",
      as: "actorDetails"
    }
  },
  {
    $project: {
      "_id": 0,
      "actorName": { $arrayElemAt:
      ↪  ["$actorDetails.primaryName", 0] },
      "averageAppearances": 1
    }
  }
])
```

2.2 Model 2

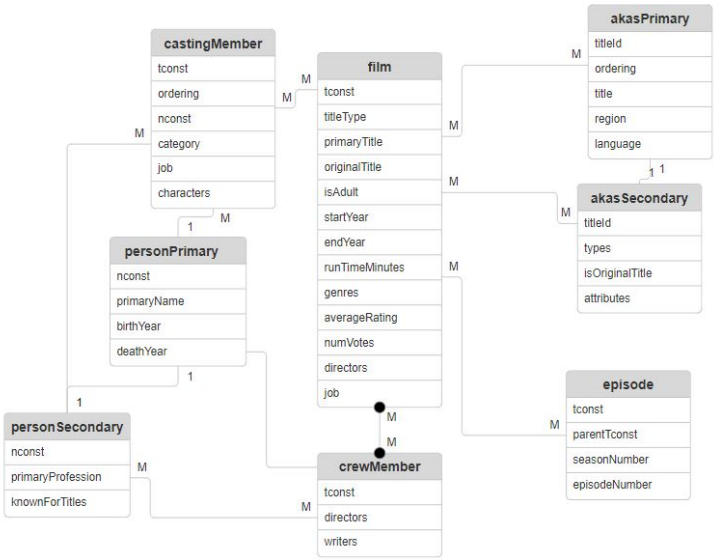

Figure 2: Entity-relationship diagram for model 2

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

```
db.film.find([
      {
        $match: {
          "titleType": "movie",
          "startYear": 1999,
          "runTimeMinutes": { $lt: 120 }
        }
      },
      {
        $addFields: {
          "ratingScore": {
            $multiply: [
              "$averageRating",
              { $log: { $add: ["$numVotes", 1] } }
            ]
          }
        }
      },
      { $sort: { "ratingScore": -1 } },
```

```
      { $limit: 10 },
      {
        $project: {
          "_id": 0,
          "tconst": 1,
          "primaryTitle": 1,
          "startYear": 1,
          "ratingScore": 1,
          "runTimeMinutes": 1
        }
      }
    ])
```

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

```
db.film.aggregate([
  {
    $match: {
      "titleType": "movie"
    }
  },
  {
    $lookup: {
      from: "akasPrimary",
      localField: "tconst",
      foreignField: "titleId",
      as: "akas"
    }
  },
  {
    $unwind: "$akas.title"
  },
  {
    $or: [
      { "primaryTitle": { $regex: "love", $options: "i" } },
      { "akas.title": { $regex: "love", $options: "i" } }
    ]
  },
  {
    $project: {
      "_id": 0,
      "tconst": 1,
      "primaryTitle": 1
```

```
    }
  }
])
```

2.2.3 Actors that played in at least 3 adult films

```
db.film.aggregate([
  {
    $match: {
      "isAdult": 1,
      "job.job": "actor"
    }
  },
  {
    $unwind: "$job"
  },
  {
    $group: {
      "_id": "$job.job",
      "count": { $sum: 1 }
    }
  },
  {
    $match: {
      "count": { $gt: 3 }
    }
  },
  {
    $lookup: {
      from: "personPrimary",
      localField: "job.nconst",
      foreignField: "nconst",
      as: "person"
    }
  },
  {
    $project: {
      "_id": 0,
      "person.name": 1,
      "job.job": 1,
      "count": 1
    }
  }
])
```

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

```
db.film.aggregate([
  {
    $unwind: "$genres"
  },
  {
    $group: {
      "_id": { "genre": "$genres", "director": "$directors"
      ↪  },
      "count": { $sum: 1 }
    }
  },
  {
    $sort: { "count": -1 }
  },
  {
    $group: {
      "_id": "$_id.genre",
      "director": { $first: "$_id.director" },
      "count": { $first: "$count" }
    }
  },
  {
    $project: {
      "_id": 0,
      "genre": "$_id",
      "director": 1,
      "count": 1
    }
  }
])

```

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

```
db.film.aggregate([
  {
    $match: {
      "titleType": "tvSeries"
    }
  },
  {
```

```
    $unwind: "$job"
  },
  {
    $match: {
      "job.job": "actor"
    }
  },
  {
    $lookup: {
      from: "personPrimary",
      localField: "job.nconst",
      foreignField: "nconst",
      as: "person"
    }
  },
  {
    $unwind: "$person.deathYear"
  },
  {
    $match: {
      $or: [
        { "person.deathYear": { $lt: "$endYear" } },
        { "person.deathYear": { $exists: true }, "endYear": {
        ↪  $exists: false } }
      ]
    }
  },
  {
    $group: {
      "_id": "$person.nconst",
      "count": { $sum: 1 }
    }
  },
  {
    $project: {
      "_id": 0,
      "count": 1
    }
  }
])

```

2.2.6 Select the top Y favorite genres of the year X

```
db.film.aggregate([
    {
      $match: {
        "titleType": "movie",
        "startYear": 1999
      }
    },
    {
      $group: {
        "_id": "$genres",
        "averageRating": { $avg: "$averageRating" }
      }
    },
    { $sort: { "averageRating": -1 } },
    { $limit: 3 },
    {
      $project: {
        "_id": 0,
        "genre": "$_id",
        "averageRating": 1
      }
    }
  ])
```

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

```
db.film.aggregate([
    {
      $match: {
        "titleType": "movie"
      }
    },
    {
      $lookup: {
        from: "akasPrimary",
        localField: "tconst",
        foreignField: "titleId",
        as: "akas"
      }
    },
    {
```

```
      $unwind: "$akas.region"
    },
    {
      $group: {
        "_id": { "region": "$akas.region", "decade": {
        ↪  $trunc: { $divide: [ "$startYear", 10 ] } } },
        "count": { $sum: 1 }
      }
    },
    {
      $group: {
        "_id": "$_id.region",
        "average": { $avg: "$count" }
      }
    },
    {
      $project: {
        "_id": 0,
        "region": "$_id",
        "average": 1
      }
    }
  ])
```

2.2.8 Average number of appearances for an actor per series

```
db.film.aggregate([
  {
    $match: {
      "titleType": "tvSeries"
    }
  },
  {
    $unwind: "$job"
  },
  {
    $match: {
      "job.job": "actor"
    }
  },
  {
    $lookup: {
      from: "episode",
      localField: "tconst",
```

```
        foreignField: "parentTconst",
        as: "episodes"
      }
    },
    {
      $unwind: "$episodes"
    },
    {
      $lookup: {
        from: "personPrimary",
        localField: "job.nconst",
        foreignField: "nconst",
        as: "person"
      }
    },
    {
      $group: {
        "_id": "$person.nconst",
        "count": { $sum: 1 }
      }
    }
    {
      $project: {
        "_id": 0,
        "person.name": 1,
        "primaryTitle": 1,
        "count": 1
      }
    }
  ])

```

3 Choice of sharding key

For this project, we propose two different types of sharding :

- Sharding on Film.job
- Sharding on Film.startYear

In the first place, we chose to shard on Film.job as our use case often requires to filter on "actor" which is an instance of the "job" attribute. In addition, we chose it because the amount of distinct jobs is known and doesn't vary too much. Consequently, the sharding storage does not scale linearly with the number of films.

For similar reasons, another extremely interesting attribute is Film.startYear which is the release year for films and the first diffusion year for series. Consequently, we decided to propose a second sharding on this attribute.

4 Communications cost model

In the tables bellow (table 1, table 2), we propose a summary of the cost models that we provide interpretation for in the conclusion (section 5). For any further information, you can refer to the google sheet we used so as to perform these cost computations.

Requête	Coût sharding 1 (job)				Coût sharding 2 (StartYear)			
	S	Shuffle	Output	#msg	S	Shuffle	Output	#msg
Ru1	100	0	1.05E+07	1.05E+11	1	0	3.16E+05	3.16E+09
Ru2	100	7.93E+13	1.03E+04	7.93E+16	100	0	1.03E+04	1.04E+07
Ru3	100	1.39E+14	3061444	6.96E+16	100	1.39E+14	3061444	6.96E+16
Ru4	100	1.39E+14	956976	1.39E+16	100	0	956976	9.57E+07
Rda1	100	1.39E+14	3061444	6.96E+15	100	1.39E+14	3061444	6.96E+15
Rda2	100	0	2360	6.15E+04	1	0	2360	5.90E+04
Rda3	100	7.93E+13	1.05E+07	7.93E+14	15.3	0	1.05E+07	1.05E+08
Rda4	100	1.12E+21	3061444	2.24E+21	100	1.12E+21	3061444	2.24E+21

Table 1: Cost Model summary for Model 1

Requête	Coût sharding 1 (job)				Coût sharding 2 (StartYear)			
	S	Shuffle	Output	#msg	S	Shuffle	Output	#msg
Ru1	100	0	1.05E+07	1.05E+11	3	0	1.05E+07	1.05E+11
Ru2	100	7.93E+13	1.03E+04	7.93E+16	100	7.93E+13	1.03E+04	7.93E+16
Ru3	24	1.39E+14	3061444	6.96E+16	100	1.39E+14	3061444	6.96E+16
Ru4	100	1.39E+14	956976	1.39E+16	100	0	956976	9.57E+07
Rda1	24	0	3061444	1.53E+08	100	306144400	3061444	1.55E+10
Rda2	100	0	2360	6.15E+04	3	0	2360	5.91E+04
Rda3	100	7.93E+13	1.05E+07	7.93E+14	100	7.93E+13	1.05E+07	7.93E+14
Rda4	24	0	3061444	6.12E+06	100	0	3061444	6.12E+06

Table 2: Cost Model summary for Model 2

5 Conclusion

In conclusion, we can see that for both models, aggregation on job is not the best solution. When aggregating on startYear, the cost is drastically lower for most queries which makes it a preferred choice for the sharding. For the model selection, the choice might not be as straight forward. In fact, depending on the more specific use case, one model or the other could be a sensible alternative.

On another aspect, we noticed that our denormalized models produce strategies that appear to be sub-optimal. In fact, while creating our denormalized models

in chapter 2, we wanted to create hybrid strategies using mostly materializations, splits, and overloads while neglecting fusions. As a consequence, queries requiring to lookup information in multiple tables are very costly. If we were to redo the project from the start, we would most certainly emphasize fusion operations to reduce the joining costs despite the fact that it increases the size of documents. An ideal strategy would be to merge the most things possible under two main collection that would be film and person.