**Chapter 2 – Denormalization**

**Epic Game Store**

**CDOF3 - LE QUERNEC Loévan / MARTIN Santiago / RINGEISSEN Léo / SALEH Julien**

Table of Contents

[Reminder of the project 2](#_Toc181830279)

[Denormalization Strategy 3](#_Toc181830280)

[Transformation Process 4](#_Toc181830281)

[1. Code of transformation 4](#_Toc181830282)

[2. Explication of the code 5](#_Toc181830283)

[Denormalized Schemas 6](#_Toc181830284)

[1. Denormalized Database 1: User-Oriented Schema 6](#_Toc181830285)

[2. Denormalized Database 2: Analyst-Oriented Schema 6](#_Toc181830286)

[3. JSON Schemas for Each Denormalized SchemA 7](#_Toc181830287)

[a) User-Oiented view 7](#_Toc181830288)

[b) Data-Analyst view 8](#_Toc181830289)

[Impact on Queries and Statistics 10](#_Toc181830290)

[1. Comparison of Query Efficiency Before and After Denormalization 10](#_Toc181830291)

[a) End-User View Queries 10](#_Toc181830292)

[b) Data Analyst View Queries 11](#_Toc181830293)

[2. Changes in document statistics 13](#_Toc181830294)

[Conclusion 14](#_Toc181830295)

# Reminder of the project

With this project, we aim to design an efficient NoSQL-based data model for a Big Data application using MongoDB. Our dataset was taken from Kaggle, and it represents the Epic Game Store. It is 268 MB in size and includes six interconnected tables: game, critic, necessary\_hardware, tweets, twitter\_accounts, and social\_networks. Each table covers a different aspect of the Epic Game Store system. This includes essential game information, user and critic reviews, social network data such as Twitter activity, and hardware requirements. These tables have complex relationships, with cardinalities that range from 1-to-1 to 1-to-many.

In our first report, we concluded that the relational schema adequately represented data. However, it was important to note that its structure created a risk for potential bottlenecks for certain queries due to multiple joins and large data volumes, especially on the analytics end of things. Taking these limitations into account, during this report we are implementing denormalization to:

* Enhance Query Performance: By embedding and merging frequently accessed data, reducing the need for joins.
* Support Scalability: Preparing for potential increases in data volume by minimizing complex query requirements.
* Improve Accessibility for Different Use Cases: Producing two new schemas
  + The first one being optimized for end-users, with quick access to essential game and social media data
  + The second one being for data analysts, offering in-depth details for comprehensive analysis.

This report will detail our approach to denormalization, including JSON schemas for both user and analyst-oriented databases, the transformation processes applied, and the impact of these choices on query efficiency and scalability.

# Denormalization Strategy

For our project, we are using different denormalization techniques to optimize our data structure. These techniques will include the embedding of frequently accessed information, which will reduce the number of joins, and the production of schemas that fit our use cases better.

We mainly applied the embedding type of denormalization to meet the differing requirements of end-users and data analysts. We are going to explain briefly what we did:

* User-Oriented Schema:
  + Denormalizations:
    - Embedded necessary\_hardware details directly in the game document to allow quick and easier access to system requirements.
    - Embedded critics as an array of reviews in the game document, enabling direct access to all reviews for a specific game.
    - Embedded social\_networks with URLs for each game to provide immediate access without separate joins.
    - Embedded tweets with a summary of recent tweets and follower count for a quick overview instead of having a complete tweet history
  + Benefits:
    - Improved Performance: The schema is optimized for fast retrieval, which is ideal for high-frequency queries typical in user-facing applications.
    - Reduced Latency: By eliminating the need for multiple joins, data is quickly accessible, enhancing the user experience.
  + Example Query Needs:
    - View reviews and ratings for a specific game.
    - Check hardware requirements without accessing a separate collection.
    - Access social media URLs directly from game documents.
* Analyst-Oriented Schema:
  + Denormalizations:
    - Embedded necessary\_hardware to make each game include its hardware requirements
    - Embedded detailed critics, social networks, and twitter information with all review details for each game
    - Embedded tweets with full details for Twitter analysis
  + Benefits:
    - Enhanced Analytical Depth: The schema offers a complete set of details, supporting data exploration and complex aggregations.
    - Scalability for Large Datasets: Although the data volume is larger, denormalization reduces the need for costly joins, making it more suitable for big data analytics.
  + Example Query Needs:
    - Calculate average ratings and engagement over time for trend analysis.
    - Analyze tweet volume and sentiment per game or genre.
    - Identify the top-rated games by publisher or trending genres based on social media interactions.

# Transformation Process

## Code of transformation

Une image contenant texte, capture d’écran

Description générée automatiquement

Une image contenant texte, capture d’écran

Description générée automatiquement

## Explication of the code

First, we start by connecting to the MongoDB instance and selecting the epic\_game\_store database, which holds our original collections. These collections include essential data about games, hardware requirements, critics' reviews, social networks, Twitter activity, and associated Twitter accounts. Once connected, we proceed with the transformation process to create two denormalized collections tailored for different use cases.

To build the user-oriented schema, we focus on embedding only the most frequently accessed information, so that users can retrieve data quickly without the need for complex joins. For each game document, we embed the hardware specifications, allowing users to view system requirements right away. Next, we embed an array of critic reviews, giving immediate access to multiple reviews within the same document. Then, we add social network links, embedding URLs and descriptions of networks related to each game. For Twitter, instead of embedding a full account history, we create a Twitter summary by pulling only the most recent 5 tweets and calculating total followers and likes, providing users with a snapshot of social media engagement. Once this data is embedded, the document is inserted into the denormalized\_games\_user collection, optimized for fast, high-frequency queries typical in user-facing applications.

In the analyst-oriented schema, we use a similar process but go into more detail to meet the needs of in-depth analysis. For each game, we embed the full details of the hardware, critic reviews, and social network links just as we do in the user-oriented schema. However, instead of summarizing the Twitter data, we embed the complete Twitter account details along with the entire tweet history, allowing analysts to explore engagement trends over time. By including all available data, we create a comprehensive dataset that enables complex aggregations and detailed analysis. The final document is then inserted into the denormalized\_games\_analyst collection, making it ideal for deeper, data-heavy queries.

Finally, the script calls both transformation functions transform\_user\_oriented() and transform\_analyst\_oriented() which populate the two separate collections. This structured approach allows us to create schemas that are tailored specifically for end-users and data analysts, enhancing the efficiency and scalability of our data model in MongoDB.

# Denormalized Schemas

## Denormalized Database 1: User-Oriented Schema

Une image contenant texte, diagramme, ligne, capture d’écran

Description générée automatiquement

## Denormalized Database 2: Analyst-Oriented Schema

Une image contenant texte, diagramme, capture d’écran, ligne

Description générée automatiquement

## JSON Schemas for Each Denormalized SchemA

### User-Oiented view

{

"game\_id": "INT",

"name": "VARCHAR",

"game\_slug": "VARCHAR",

"price": "DECIMAL",

"release\_date": "DATE",

"platform": "VARCHAR",

"description": "TEXT",

"developer": "VARCHAR",

"publisher": "VARCHAR",

"genres": "VARCHAR",

"necessary\_hardware": {

"hardware\_id": "INT",

"operating\_system": "VARCHAR",

"processor": "VARCHAR",

"memory": "VARCHAR",

"graphics": "VARCHAR",

"fk\_game\_id": "INT"

},

"average\_rating": "DECIMAL",

"total\_reviews": "INT",

"critics": [

{

"critic\_id": "INT",

"company": "VARCHAR",

"author": "VARCHAR",

"rating": "DECIMAL",

"comment": "TEXT",

"date": "DATE",

"top\_critic": "BOOLEAN",

"fk\_game\_id": "INT"

}

],

"social\_networks": [

{

"social\_network\_id": "INT",

"description": "VARCHAR",

"url": "VARCHAR",

"fk\_game\_id": "INT"

}

],

"twitter\_summary": {

"total\_followers": "INT",

"total\_likes": "INT",

"recent\_tweets": [

{

"tweet\_id": "INT",

"text": "TEXT",

"url\_media": "VARCHAR",

"likes": "INT",

"retweets": "INT",

"timestamp": "DATETIME",

"fk\_twitter\_account\_id": "INT"

}

]

}

}

### Data-Analyst view

{

"game\_id": "INT",

"name": "VARCHAR",

"game\_slug": "VARCHAR",

"price": "DECIMAL",

"release\_date": "DATE",

"platform": "VARCHAR",

"description": "TEXT",

"developer": "VARCHAR",

"publisher": "VARCHAR",

"genres": "VARCHAR",

"necessary\_hardware": {

"hardware\_id": "INT",

"operating\_system": "VARCHAR",

"processor": "VARCHAR",

"memory": "VARCHAR",

"graphics": "VARCHAR",

"fk\_game\_id": "INT"

},

"critics": [

{

"critic\_id": "INT",

"company": "VARCHAR",

"author": "VARCHAR",

"rating": "DECIMAL",

"comment": "TEXT",

"date": "DATE",

"top\_critic": "BOOLEAN",

"fk\_game\_id": "INT"

}

],

"social\_networks": [

{

"social\_network\_id": "INT",

"description": "VARCHAR",

"url": "VARCHAR",

"fk\_game\_id": "INT"

}

],

"twitter\_accounts": [

{

"twitter\_account\_id": "INT",

"name": "VARCHAR",

"username": "VARCHAR",

"bio": "TEXT",

"location": "VARCHAR",

"website": "VARCHAR",

"join\_date": "DATE",

"followers": "INT",

"following": "INT",

"fk\_game\_id": "INT",

"tweets": [

{

"tweet\_id": "INT",

"text": "TEXT",

"url\_media": "VARCHAR",

"quantity\_likes": "INT",

"quantity\_retweets": "INT",

"quantity\_quotes": "INT",

"quantity\_replies": "INT",

"timestamp": "DATETIME",

"fk\_twitter\_account\_id": "INT"

}

]

}

]

}

# Impact on Queries and Statistics

## Comparison of Query Efficiency Before and After Denormalization

The denormalization process has significantly improved query efficiency, especially for high-frequency user queries and complex analytical queries. As we’ve stated multiple times in this report, in the original normalized schema multiple joins were required to retrieve related data, resulting in increased query execution times and computational costs. This was especially noticeable in queries that required accessing reviews, social media information, or hardware requirements for each game.

### End-User View Queries

* View Reviews for a Specific Game
  + Targeted Tables and Attributes:
    - games: game\_id, name, game\_slug
    - critics: author, rating, comment, date
  + Joins to be Performed:
    - None (reviews are embedded in critics array within the game document)
  + Attribute Filters:
    - Filter on game\_slug to identify the specific game (game\_slug = 'some\_game\_slug')
  + Projections:
    - game.name, critics.author, critics.rating, critics.comment, critics.date
  + Aggregates:
    - None (directly retrieves all reviews for the specified game)
* Find Hardware Requirements for a Game
  + Targeted Tables and Attributes:
    - games: game\_id, name, game\_slug
    - necessary\_hardware: operating\_system, processor, memory, graphics
  + Joins to be Performed:
    - None (hardware requirements are embedded in necessary\_hardware array within the game document)
  + Attribute Filters:
    - Filter on game\_slug to identify the specific game (game\_slug = 'some\_game\_slug')
  + Projections:
    - game.name, necessary\_hardware.operating\_system, necessary\_hardware.processor, necessary\_hardware.memory, necessary\_hardware.graphics
  + Aggregates:
    - None (directly retrieves hardware details for the specified game)
* Search for Social Networks for a Game
  + Targeted Tables and Attributes:
    - games: game\_id, name, game\_slug
    - social\_networks: url, description
  + Joins to be Performed:
    - None (social network URLs are embedded in social\_networks array within the game document)
  + Attribute Filters:
    - Filter on game\_slug to identify the specific game (game\_slug = 'some\_game\_slug')
  + Projections:
    - game.name, social\_networks.url, social\_networks.description
  + Aggregates:
    - None (directly retrieves social network URLs and descriptions for the specified game)
* View Tweets Related to a Game
  + Targeted Tables and Attributes:
    - games: game\_id, name, game\_slug
    - twitter\_summary: recent\_tweets
  + Joins to be Performed:
    - None (recent tweets are embedded in twitter\_summary array within the game document)
  + Attribute Filters:
    - Filter on game\_slug to identify the specific game (game\_slug = 'some\_game\_slug')
    - Optionally filter on recent\_tweets.timestamp for time-based filtering (e.g., last 24 hours)
  + Projections:
    - game.name, twitter\_summary.recent\_tweets
  + Aggregates:
    - None (directly retrieves recent tweets related to the game)

### Data Analyst View Queries

* Average Rating for Each Game Over Time
  + Targeted Tables and Attributes:
    - games: game\_id, name
    - critics: rating, date
  + Joins to be Performed:
    - None (reviews are embedded in critics array within the game document)
  + Attribute Filters:
    - Filter critics.date to only include recent reviews (e.g., critics.date >= 'some\_date')
  + Projections:
    - game.name, MONTH(critics.date)
  + Aggregates:
    - AVG(critics.rating): Calculates the average rating for each month
    - COUNT(critics.rating): Counts the number of reviews per month
* Identify Most Active Games on Twitter
  + Targeted Tables and Attributes:
    - games: game\_id, name
    - twitter\_accounts: tweets, timestamp
  + Joins to be Performed:
    - None (full tweet history is embedded in twitter\_accounts.tweets array within the game document)
  + Attribute Filters:
    - Filter twitter\_accounts.tweets.timestamp for recent activity (e.g., tweets.timestamp >= 'some\_date')
  + Projections:
    - game.name
  + Aggregates:
    - COUNT(tweets.id): Calculates the total number of tweets per game to determine the most active games
* Top 5 Publishers by Average Game Rating
  + Targeted Tables and Attributes:
    - games: game\_id, name, publisher
    - critics: rating
  + Joins to be Performed:
    - None (ratings are embedded in critics array within the game document)
  + Attribute Filters:
    - Filter games to only include those with at least 10 reviews (e.g., COUNT(critics.rating) >= 10)
  + Projections:
    - game.publisher
  + Aggregates:
    - AVG(critics.rating): Calculates the average rating for each publisher
    - COUNT(critics.rating): Ensures only publishers with enough reviews are included
    - LIMIT 5: Retrieves the top 5 publishers by rating
* Trending Genres Based on Twitter Activity and Reviews
  + Targeted Tables and Attributes:
    - games: game\_id, name, genres
    - critics: rating
    - twitter\_accounts: tweets
  + Joins to be Performed:
    - None (both ratings and tweets are embedded within the game document in the denormalized schema)
  + Attribute Filters:
    - Filter games with high ratings (e.g., AVG(critics.rating) > 80%) to focus on highly-rated games.
    - Filter tweets by timestamp to focus on recent activity (e.g., tweets.timestamp >= 'some\_date').
  + Projections:
    - game.genres: Retrieves genres for each game.
  + Aggregates:
    - COUNT(tweets.id): Counts the number of tweets per genre to gauge activity and engagement on social media.
    - AVG(critics.rating): Calculates the average rating per genre to focus only on genres with high reviews.

## Changes in document statistics

The denormalized schemas have led to changes in document counts and the structure of embedded documents within collections. Here are the document statistics and embedded document details post-denormalization:

* User-Oriented Schema:
  + Total Games: 915 documents, each containing embedded data for hardware requirements, critics, social networks, and a Twitter summary.
  + Embedded Documents per Game:
    - Necessary Hardware: 1 document per game, with an average of 2-3 fields embedded.
    - Critics: Average of 20 reviews per game, each with fields such as rating, author, and comment.
    - Social Networks: 1-3 social network links per game, providing direct access to URLs.
    - Twitter Summary: Summarized with key metrics and 3-5 recent tweets per game.
* Analyst-Oriented Schema:
  + Total Games: 915 documents, each embedding complete datasets for critics and Twitter accounts.
  + Embedded Documents per Game:
    - Necessary Hardware: 1 document per game, as in the user-oriented schema.
    - Critics: Full list of reviews for each game, supporting comprehensive analysis.
    - Social Networks: Complete social network links, as in the user-oriented schema.
    - Twitter Accounts: Each game includes 1-2 Twitter accounts, with a full history of tweets (averaging around 50 tweets per account).

With this new data structure, we have larger document sizes in the analyst-oriented schema, especially for games with large Twitter activity or numerous reviews. However, the benefits in query efficiency and scalability are more important than the increase in document size, as the data model now supports both of our use cases, user-friendly queries and in-depth analysis, with less significant downsides.

# Conclusion

Our denormalization of the Epic Game Store dataset has provided important improvements in both query performance and scalability, satisfying the goals that we established at the beginning of this report. By embedding frequently accessed data and creating distinct schemas for our use cases of end-users and data analysts, we have achieved a data model that meets diverse needs without sacrificing efficiency.

For end-users, the denormalized and more user-oriented schema allows for quicker access to essential information, reducing latency and enhancing the responsiveness of user-facing applications. By eliminating the need for multiple joins, this new schema allows us to have a faster retrieval of game details, hardware requirements, reviews, and social media links, improving the overall user experience.

For data analysts, the analyst-oriented schema gives a comprehensive data structure that supports advanced analytics, trend analysis, and sentiment tracking. The inclusion of full historical data for reviews and tweets allows for deeper analysis and improves complex aggregations, making the schema scalable and adaptable to larger data volumes.

Overall, our denormalization strategy has successfully dealt with the limitations of the normalized schema, transforming the dataset into a robust, efficient, and scalable NoSQL data model.