Advanced Databases

Project Report

for

TV Shows Database

Under the Guidance of:

Prof. Frank Hefter

Prof. Dr. Barbara Sprick

Submitted By:

Team RED

Adimulam Hari Har Nath, Anirudh

Pandey, Yash

Pokarne, Vishal

Rane, Manjiri

Thakur, Prajakta

June 18, 2020

Contents

List of Figures

List of Tables

Introduction

TV Shows have been one of the critical aspects of our entertainment. A commitment to a TV show is much larger than that of the movies; Largely because a good TV show generally runs from a minimum of 4 seasons through somewhere around 10 or even 10+ seasons, with each season containing anywhere from 10 to 24 episodes approximately. So, we invest a great amount of time in watching the TV shows.

With the rise in the streaming wars between several streaming content providers, such as Netflix, HBO, Amazon Prime, Disney+, Hotstar, Hulu, etc., the users are forced to pay for each service individually just to access quality content.

In this project, we are introducing our application, “TV Shows database”, where users can sign up to be up to date with all the TV shows from various streaming medias. Users can follow other users, and get recommendations based on the watching history of other users whom they follow. The existing users can share their referral codes to new users to get reward bonus points. There is a premium users option - where the users who are paying for the premium services get an access to create a **Watch Party**, a cool, nifty feature which comes in handy in difficult times like the COVID-19 pandemic, where they can create a temporary room and invite their friends to watch the particular TV shows together.

Organization

Work Breakdown Structure

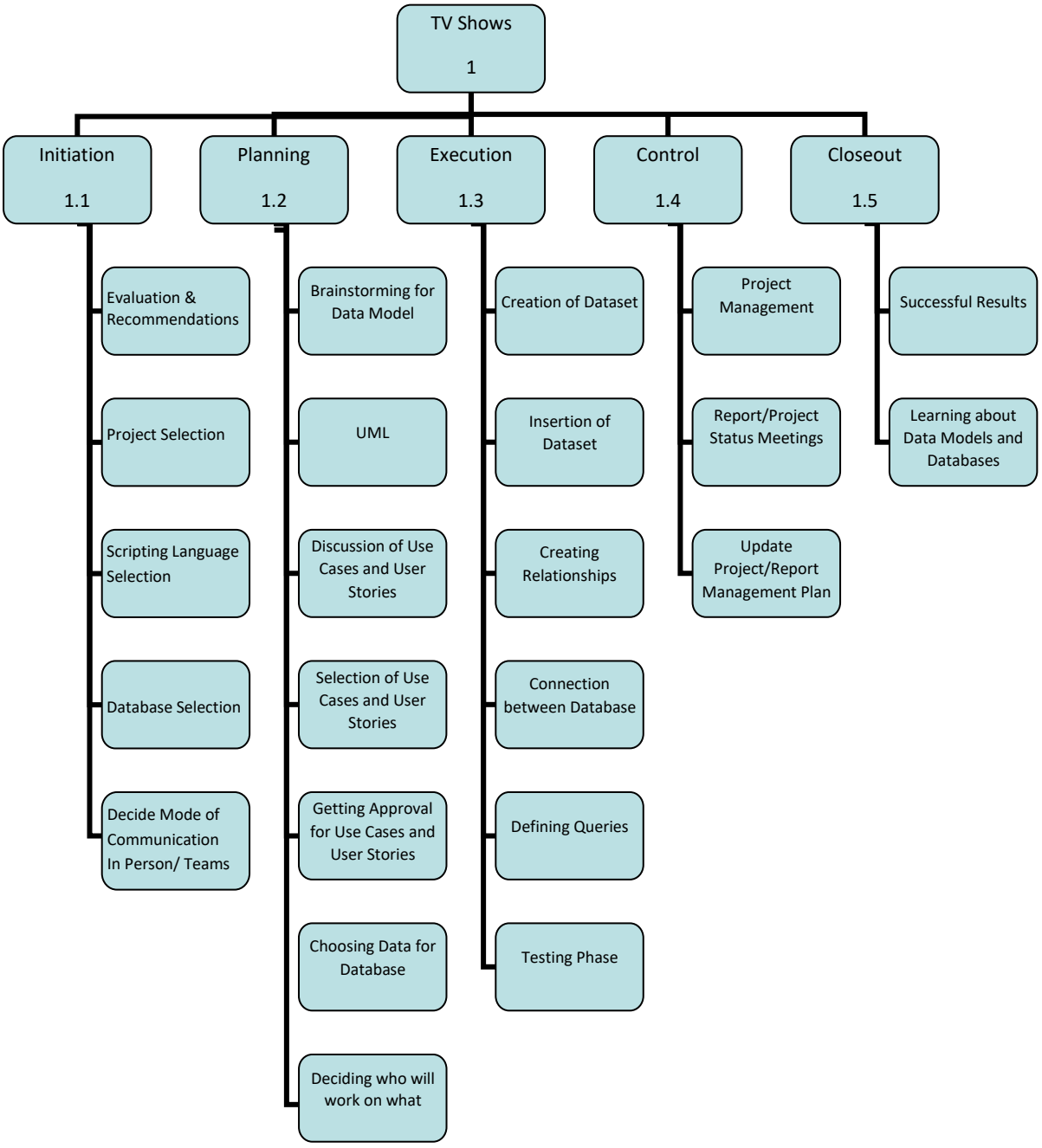


Figure 1. Work Breakdown Structure

The TV shows database management system is entirely divided into five phases while development. The phases are Initiation, Planning, Execution, Control and Closeout.

Initiation: This was the initial stage of our project where team members came up ideas and recommendations for projection selection. After spending valuable amount of time on Project Selection, we decided over for database selection and scripting language selection with

research based on requirements. In times like Covid19, it was important for us to decide the mode of communication for every discussion and project meeting.

Planning: Planning started with brainstorming for Data models and developing UML for the same. Every group member was given a task to come up with few User Stories and use cases. After discussing with team members, we decided over few use cases to get approved from Professor. Meanwhile, we were working on choosing Database for the same. After getting approval for Use cases we divided our work among team member.

Execution: With Execution, we started creating our Data set and thus for the insertion of same after creation. We have divided our use cases in such a way that one person can work on minimum two database and maximum three database. In this phase, everyone was responsible to work on setting up suitable database for their use case and share the work, among others. Creating queries for the use cases came after this for the concerned person. After defining queries, we carried out a testing phase for our project.

Control: Every task should be completed along a given timeline following guidelines and giving desirable results was at highest priority. For achieving this, project management was followed according to provided timeline by professor. Updating Project Status plan alongside with report was carried out by team members at every step of project completion.

Closeout: After the testing of queries and successful results for the same in our TV shows database.

Minutes of the Meeting

Table 1. Meeting 1

Meeting: 01		
Date: 18th May 2020	**Time:** 11:00	**Location:** MS Teams
Topic of discussion	Brainstorming of project topic	
Meeting called by	Yash Pandey	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Project topic decided	

Table 2. Meeting 2

Meeting: 02		
Date: 24th May 2020	**Time:** 13:00	**Location:** MS Teams
Topic of discussion	Selection of databases and scripting language	
Meeting called by	Prajakta Thakur	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Databases and scripting languages finalized	

Table 3. Meeting 3

Meeting: 03		
Date: 25th May 2020	**Time:** 19:00	**Location:** MS Teams
Topic of discussion	Use cases and user stories	
Meeting called by	Anirudh Adimulam	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Finalized use cases and user stories for draft 1	

Table 4. Meeting 4

Meeting: 04		
Date: 28th May 2020	**Time:** 18:00	**Location:** MS Teams
Topic of discussion	Changes in use cases and user stories	
Meeting called by	Manjiri Rane	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Changes as per feedback	

Table 5. Meeting 5

Meeting: 05		
Date: 30th May 2020	**Time:** 14:00	**Location:** MS Teams
Topic of discussion	Preparing the dataset	
Meeting called by	Vishal Pokarne	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Prepared the CSV file	

Table 6. Meeting 6

Meeting: 06		
Date: 31st May 2020	**Time:** 19:00	**Location:** MS Teams
Topic of discussion	Insertion of data	
Meeting called by	Yash Pandey	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Data inserted	

Table 7. Meeting 7

Meeting: 07		
Date: 2nd June 2020	**Time:** 19:00	**Location:** MS Teams
Topic of discussion	Discussion and planning	
Meeting called by	Prajakta Thakur	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Planned further steps	

Table 8. Meeting 8

Meeting: 08		
Date: 7th June 2020	**Time:** 17:00	**Location:** MS Teams
Topic of discussion	Status meeting	
Meeting called by	Vishal Pokarne	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Each team member shared their status	

Table 9. Meeting 9

Meeting: 09		
Date: 10th June 2020	**Time:** 13:00	**Location:** MS Teams
Topic of discussion	Draft 3 report	
Meeting called by	Anirudh Adimulam	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Almost done with the draft 3 report	

Table 10. Meeting 10

Meeting: 10		
Date: 14th June 2020	**Time:** 11:00	**Location:** MS Teams
Topic of discussion	Writing queries	
Meeting called by	Manjiri Rane	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Wrote queries for the use cases	

Table 11. Meeting 11

Meeting: 11		
Date: 16th June 2020	**Time:** 12:00	**Location:** MS Teams
Topic of discussion	Final report	
Meeting called by	Yash Pandey	
Attendees	Anirudh Adimulam, Yash Pandey, Vishal Pokarne, Manjiri Rane, Prajakta Thakur	
Conclusion	Worked on the final report	

Roles and Responsibilities

Table 12. Roles and Responsibilities

Tasks	**Project Team Members**				
	Anirudh	**Yash**	**Vishal**	**Manjiri**	**Prajakta**
Brainstorming	R	R	R	R	R
Database setup and data import	R	R	R	R	R
Requirement Gathering	R	R	R	R	R
UML	I	I	A	I	R
Data Schema	R	C	C	C	C
User Stories	R	R	R	R	R
Use Case 01	I	A	R	I	I
Use Case 02	I	I	I	R	I
Use Case 03	I	R	I	I	I
Use Case 04	I	I	I	I	R
Use Case 05	R	I	I	I	I
Documentation	A	R	R	A	A

R	Responsible
A	Accountable
C	Consulted
I	Informed

User Stories

US1. Referral Bonus Points

Description: As a free or premium user, I can refer this application to new users using a unique referral code assigned to me. I can share my referral code with new users which will benefit me and them with some additional bonus points, which can be used to extend the validity of the premium subscription.

US2. Searching a TV Show based on rating and the genre

Description: As a free or premium user, I can search for a TV show based on genre as well as ratings. The main condition is that searching based on both the criteria, i.e. the list of TV shows of a specific genre and the shows having rating greater than or equal to the desired rating will be displayed as a result of my search. Every time I log in, the search performed during that session will be stored. Every search will have my username as well as the date and timestamp at which the search was performed. Now that my search history is being stored, I can manage it as well i.e. I can fetch my search history and even delete a search if I wish to.

US3. Getting a recommendation of TV shows which are related to the Show that the user has watched

Description: As a user or a premium user, I will search for a TV show to watch. When I watch a TV show, I will get recommendations for all the shows that are connected to the show that I have watched. I can like and unlike a TV show that I have watched, and I can also dislike or remove a show from the list of my disliked shows. If I wish, then I can remove a TV show from the list of my watched shows.

US4. Creating a Watch Party

Description: As a user I can upgrade myself to premium user. As a premium user, I can create a watch party, which essentially is a temporary room, where I can watch a TV show with other users. When I create a watch party, an invitation code is generated, which can be shared with other users who are my friends. Now other users can see whose watch parties are available among his/her friends and if they want to join my watch party, they can use my invitation code. This will be a good feature during the current times of COVID-19 when everyone is practicing social distancing.

US5. TV Shows and User management

Description: As an Administrator I can have complete access to create, update a user account, and delete a user account if it is no more operational. As an Administrator I can add a new TV shows to the list or delete an existing show if it is no more watched by any user. As an Administrator I will have access to see most liked and disliked shows by users. If a show is disliked by most of the users later, it will be deleted.

Database Structure

UML Diagram

Figure 2. UML Diagram

Database Schema

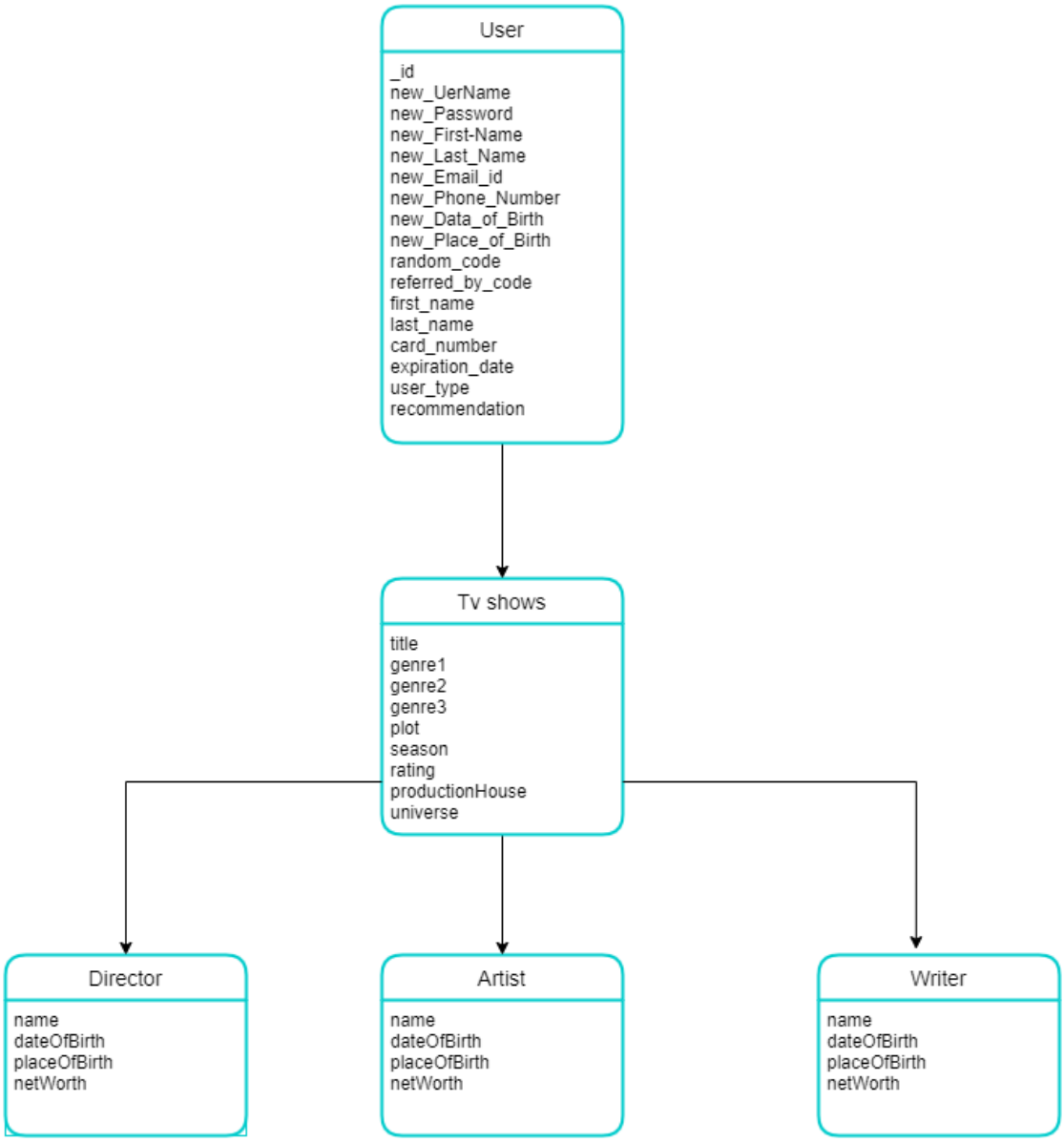


Figure 3. Database Schema

Databases Selection

MongoDB

MongoDB is the most popular NoSQL database today, empowering users to query, manipulate and find interesting insights from their data. MongoDB has grown from being just a JSON data store to become the most popular NoSQL database solution with efficient data manipulation and administration capabilities. The sharding and aggregation framework, coupled with document validations, fine-grained locking, a mature ecosystem of tools and a

vibrant community of users are some of the key reasons why MongoDB is the go-to database for many.

In MongoDB we are storing data for our database entities in the form of documents like Users, Administrator, Reward.

Why MongoDB?

MongoDB's horizontal, scale-out architecture can support huge volumes of both data and traffic. We are storing User details both premium and normal user, Administrator details, Reward details in MongoDB for the similar reason. Document databases allow embedding of documents to describe nested structures and easily tolerate variations in data in generations of documents.

Neo4j

Neo4j is an open source, distributed data store used to model graph problems. It departs from the traditional nomenclature of database technologies, in which entities are stored in schema-less, entity-like structures called **nodes**, which are connected to other nodes via relationships or edges

In neo4j we are storing application data of entire TV shows to maintain relationships between nodes. Along with the TV shows data we are storing User relationships as friends or referee for referral code.

Why Neo4J?

The performance of Neo4j does not drop, even though data queries increase exponentially. Graph databases respond to inquiries by updating the node and the relationships of that search and not the whole of the complete graph which optimizes the process. We were looking for flexibility and scalability as when the TV shows data increases. Graph databases contribute a lot in this regard because when needs increase, **the possibilities of adding more nodes and relationships to an existing graph are huge**.

Redis

Redis, which stands for Remote Dictionary Server, is a fast, open-source, in-memory key-value data store for use as a database, cache, message broker, and queue. Redis is a popular choice for caching, session management, gaming, leader boards, real-time analytics, geospatial, ride-hailing, chat/messaging, media streaming, and pub/sub apps. Redis maps keys to types of values. Difference between Redis to other database is store not only strings but also abstract data types.

In Redis, we are storing details of temporary room created by Premium user also called as Watch Party. We are also storing search history of user in Redis.

Why Redis?

Redis supports Pub/ Sub with pattern matching and a variety of data structure such as lists, sorted sets and hashes. This allows Redis to support high performance real-time comment streams or watch parties and server intercommunication. As the temporary room details and search history will be temporary, we thought of storing it in Redis. Redis gives high-performing cache system. As our users, are required to retrieve data from the database within a second. if not, it will affect our business. So, we have used Redis database.

Data Models

Graph Data Model

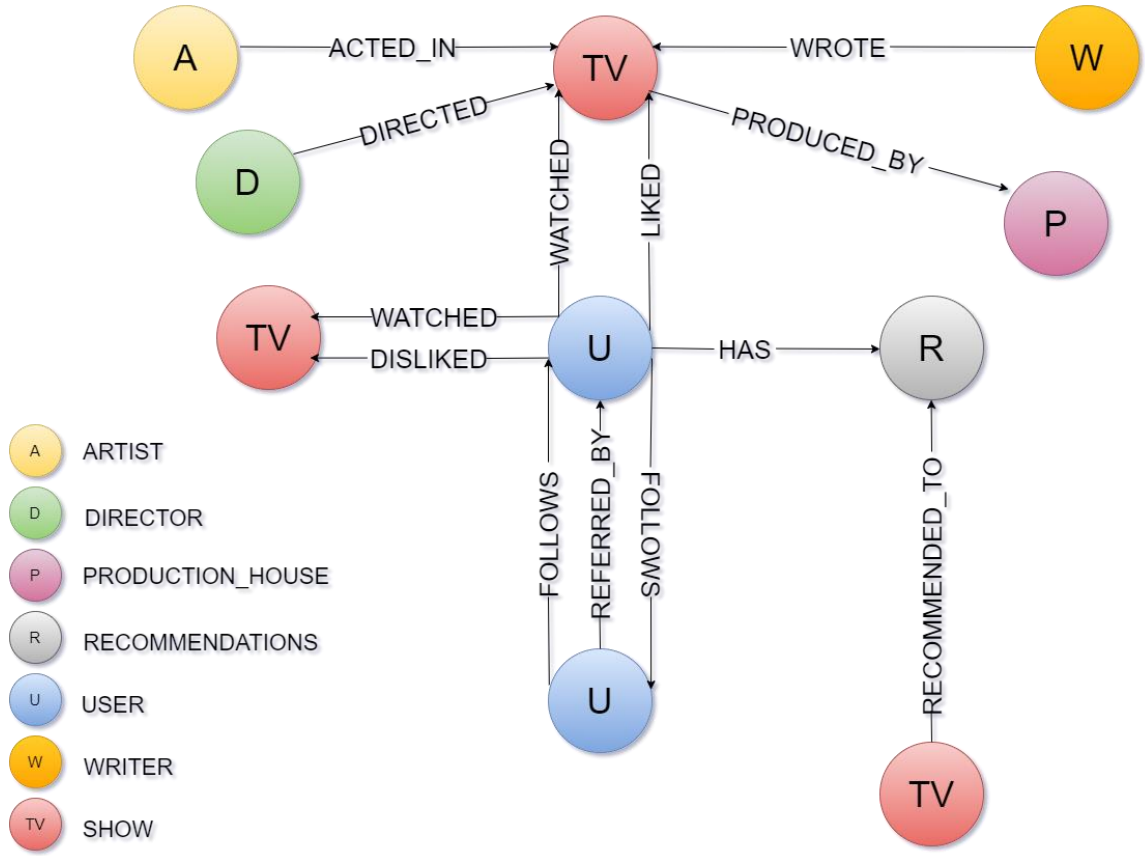


Figure 4. Graph Data Model

Document Data Model

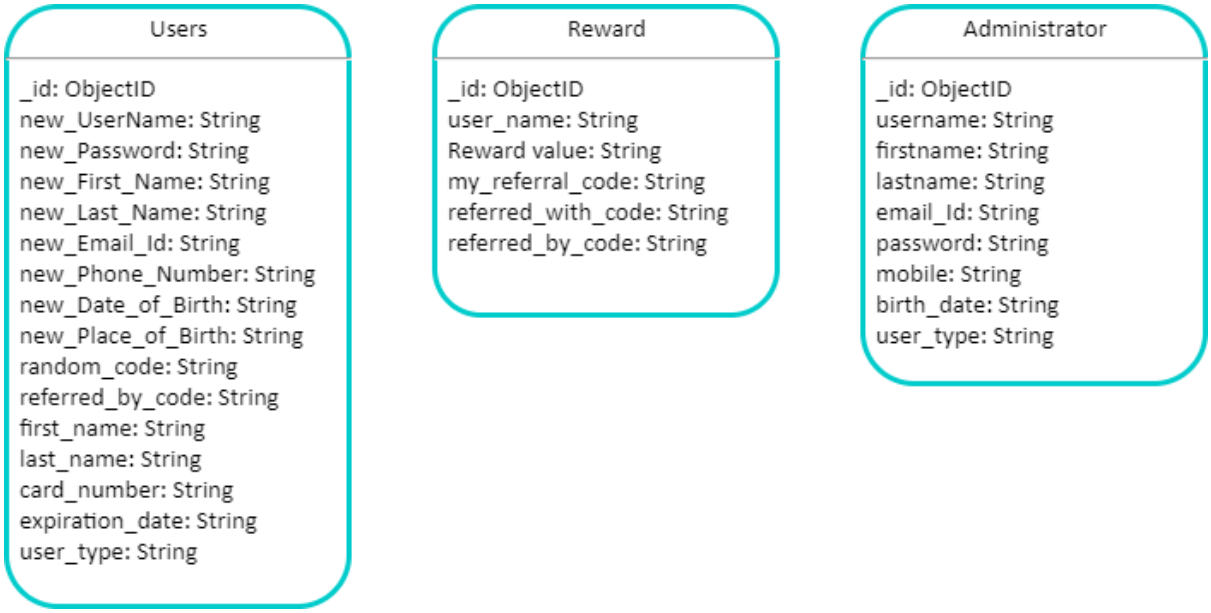


Figure 5. Document Data Model

Key-value Data Model

Redis Hash:

User search history with genre and rating as fields:

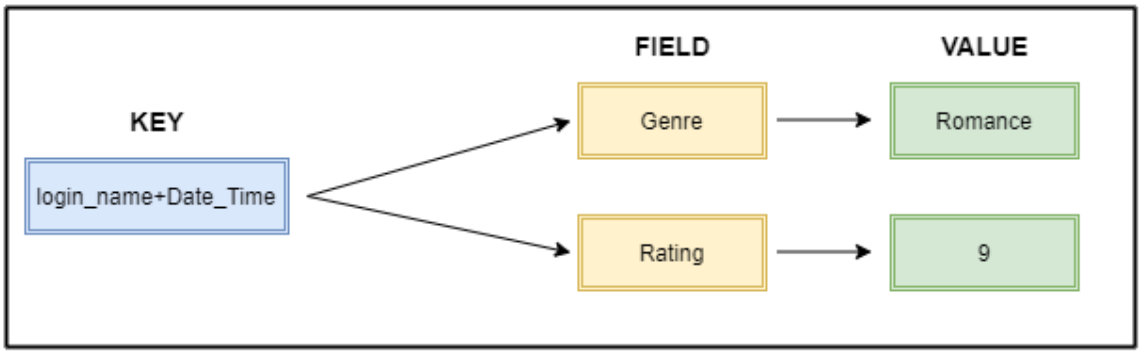


Figure 6. Key-value Data Model 1

Watch room created by user:

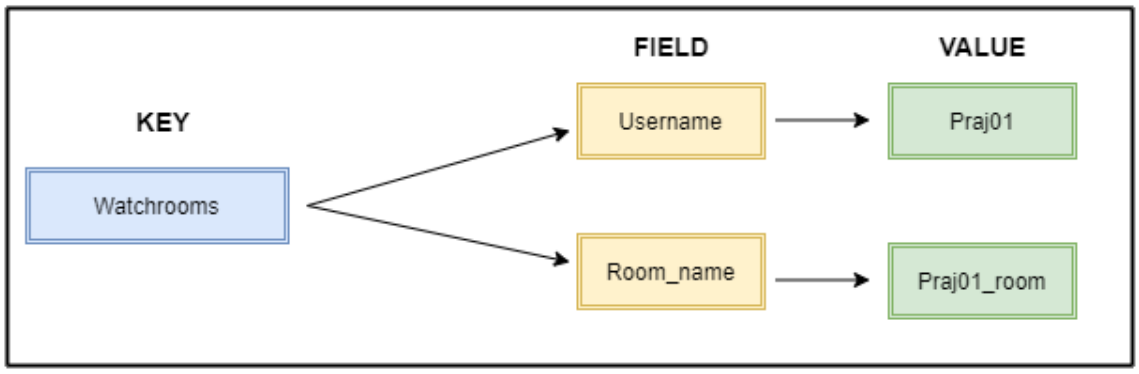


Figure 7. Key-value Data Model 2

Room details with assigned invitation code:

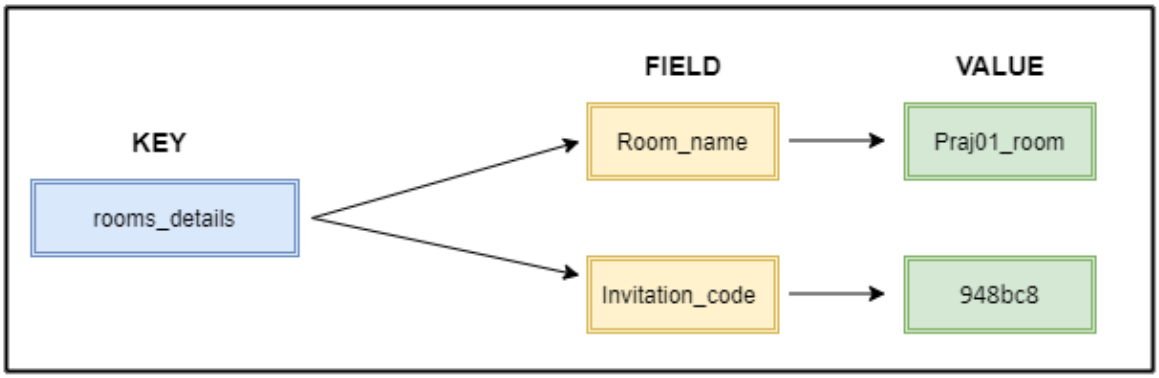


Figure 8. Key-value Data Model 3

Redis List

List of members joining a watchroom:

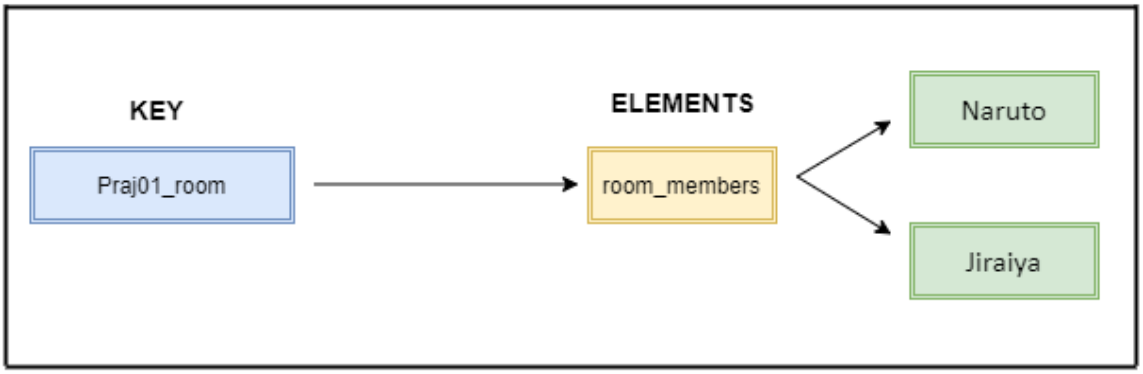


Figure 9. Key-value Data Model 4

Use Cases

Use Case 1

Table 13. Use Case 1

Use Case	Referral Bonus Points
Participating actors	Two users, one already existing (the referrer) and one new (the referee)
Flow of events	1. First user creates an account. 2. He gives his referral code to second user when the second user wants to create an account. 3. The second user enters the referral code and the username of the first user correctly during the sign-up process. 4. Both the users get 50 bonus points in their account.
Entry Condition	A user wants referral bonus points.
Exit Condition	Positive: The user receives the bonus points. OR Negative: The user does not receive the bonus points.
Use of databases	1. **MongoDB:** To store the referral codes and the reward points of each user. 2. **Neo4j:** To create the relationships between the referee and the referrer.

Use Case 2

Table 14. Use Case 2

Use Case	Search based on rating, genre and storing the search history
Actors	User and Premium user
Flow of events	1. User will enter their genre preference. 2. User will enter the rating. 3. The list of TV shows from the genre will be displayed. 4. The ratings of the TV shows displayed will be greater than or equal to the rating entered by user. 5. The search performed by the user will be stored. 6. User can view his/ her search history records stored as hash in Redis. 7. The search history records can also be deleted by the user.
Entry Condition	Users will select the rating and genre-based search option.
Exit Condition	Positive: Users will get the shows based upon his preferred genre and ratings. User will view the list of search history records. OR Negative: Users will not receive any shows based upon the combination of his search.
Use of databases	1. **Neo4j:** The list of shows and their ratings will be fetched. 2. **Redis:** the search history of the user every time he performs a search.

Use Case 3

Table 15. Use Case 3

Use Case	Get recommendations based on the shows watched
Actors	User and Premium user
Flow of events	1. The user searches for a TV show from a list or by the show title. 2. The user selects a TV show and watches it. 3. When the user watches a TV show, a recommendation list is created for him/ her. 4. Shows which are related to the TV show that the user has watched are fetched. 5. If there are similar shows, then they are added in the recommendations list of the user. 6. The user can see the list of recommended shows to watch and watch a show. 7. The user can like or unlike a TV show from the shows he/ she has watched. 8. The user can dislike a TV show or remove the TV show from their list of disliked shows. 9. The user can also remove a TV show from the list of watched shows.
Entry Condition	The user wants to watch a show and get recommendations based on it.
Exit Condition	The user received the acknowledgement of the shows recommended.
Use of databases	**Neo4J** is used to fetch the TV shows to watch. After a show is watched by the user, similar shows are retrieved from Neo4j for recommendation.

Use Case 4

Table 16. Use Case 4

Use Case	Creating a Watch Party
Actors	User and Premium user
Flow of events	1. User can Upgrade to Premium User by using sufficient cash points or through payment. 2. Premium User can create a temporary room called as watch party. 3. Creation of watch Party will give user invitation code. 4. The user shares this invitation code with his/her friends. 5. The friends can look for available watch parties from his/her friends list 6. The friend enters watch party using specific invitation code shared with by his/her friend.
Entry Condition	The user wants to create watch party and watch show together with friends.
Exit Condition	User who created watch party is hosting watch party and the users who are friends enters watch party with invitation code shared with them.
Use of databases	1. **Redis**: a. The room created is temporary and invitation code of watch party is saved in Redis. b. We are also using Redis to store all users who are there watch party 2. **MongoDB**: a. -We are taking user login details from MongoDB where the type of user as “Premium” or “NonPremium” is checked. b. -For using cash points, while upgrading normal user to premium the data is fetched from MongoDB 3. **Neo4J**: We are storing relationship between users in Neo4j, so that users can find out his/her friends and who they follow before joining a watch party.

Use Case 5

Table 17. Use Case 5

Use Case	TV Shows, User management and Analytics
Actor	Administrator
Flow of events	1. Administrator will get a request for new account. 2. After receiving user information, Administrator will store user information into database. 3. If administrator get request from user regarding deletion or update of account administrator will delete or update the account. 4. Administrator can add new TV show if there is any new show released. Administrator can update any existing TV show if there is any new episode released. 5. Administrator will have statistics of most and least liked and watched show of user satisfaction in term or regular update in tv shows.
Entry Condition	Administrator wants to create update or delete any user or TV show.
Exit Condition	Administrator will receive a confirmation after adding or deleting.
Use of databases	1. **MongoDB:** User information is stored in Mongo DB. 2. **Neo4J:** TV shows and their relations are stored in Neo4j.

Queries

US1 Queries: Referral bonus points

Creating the document of the new user (who has used the referral code) after verification of the referrer:

```
collection_new_user.insert_many(
        [
            {
                "new_UserName": Vishal,
                "new_Password": password,
                "new_First_Name": Vishal,
                "new_Last_Name": Pokarne,
                "new_Email_id": vishalpokarne@gmail.com,
                "new_Phone_Number": 09039944839,
                "new_Date_of_Birth": 03.08.1994,
                "new_Place_of_Birth": Aurangabad,
                "random_code": Vishal09hty3,
                "referred_by_code": SashaaU7tRE8
            },
        ]
    )
```

Fetching the details of the referee (who has used the referral code) for further operations:

```
data_referree = collection_reward.find_one({"my_referral_code"
: Vishal09hty3})
```

Fetching the details of the referrer (who has given the referral code) for further operations:

```
data_referrer = collection_reward.find_one({"my_referral_code"
: SashaaU7tRE8})
```

Updating the cash reward and referrer details of the referee:

```
collection_reward.replace_one({"_id": 34},
{"user_name": Vishal,
"Reward value": 50,
"my_referral_code": Vishal09hty3,
"referred_by_code": SashaaU7tRE8})
```

Updating the cash reward details of the referrer (who has given the referral code):

```
collection_reward.replace_one({"_id": 38},
{"user_name": Sashaa,
"Reward value": 50,
"my_referral_code": SashaaU7tRE8,
"referred_with_code": SashaaU7tRE8})
```

Creating the nodes of the referrer and referee, and creating the relationship between the two:

```
MERGE (u1:USER{username:"Google"})
MERGE (u2:USER{username:"Pixel"})
MERGE (u1) -[:REFERRED_BY{referralCode:"Phone"}] -> (u2)
RETURN (u1) -- (u2)
```

Creating the node of the user if they do not have any referrer:

```
MERGE (u1:USER{username:"Pixel"})
```

US2 Queries: Searching a TV Show based on rating and the genre

Searching tv shows from Neo4j based upon user's genre and rating preference:

```
MATCH (p:SHOW)
WHERE (p.genre1="Drama" OR p.genre2="Drama" OR
p.genre3="Drama"  AND p.IMDBRating >"5")
RETURN p.title, p.rating
```

Storing the search history in Redis:

```
hset Manjiri Genre Drama
hset Manjiri Rating 9
```

Deleting the search history:

```
del Manjiri
```

US3 Queries: Getting recommendations of TV Shows related to the watched show

Fetch TV show titles of all the shows in the database:

```
MATCH (tv:SHOW) RETURN tv.title
```

Fetch the title of TV show "Arrow":

```
MATCH (tv:SHOW {title: "Arrow"}) RETURN tv.title
```

Generate a recommendations node for the user "yash":

```
MERGE (r:RECOMMENDATIONS{name: "Recommendations for you",
owner: "yash"})
MERGE (u:USER{username: "yash"})
MERGE (u)-[:HAS]->(r)
```

Remove TV show "Arrow" from recommendations of "yash" before watching it:

```
MATCH (tv:SHOW {title: "Arrow"})
MATCH (r:RECOMMENDATIONS{owner: "yash"})
MATCH (tv)-[rel:RECOMMENDED_TO]->(r)
DELETE rel
```

Yash will watch TV show "Arrow" and universe "Arrowverse" is returned to fetch the similar shows for recommendations:

```
MATCH (u:USER {username: "yash"})
MATCH (tv:SHOW {title: "Arrow"})
MERGE (u)-[:WATCHED]->(tv)
RETURN tv.universe
```

Recommend TV shows related to "Arrow" which user "yash" has watched:

```
MATCH (tv:SHOW {universe: "Arrowverse"})
MATCH (r:RECOMMENDATIONS {name: "Recommendations for you",
owner: "yash"})--(u:USER {username: "yash"})
WHERE NOT (u)-[:WATCHED]->(tv)
MERGE (tv)-[:RECOMMENDED_TO]->(r)
```

Fetch the titles of all the shows that "yash" has watched:

```
MATCH (u:USER {username: "yash"})-[:WATCHED]->(tv:SHOW)

RETURN tv.title
```

User “yash” will like TV show “Arrow” which he has watched:

```
MATCH (u:USER {username: "yash"})
MATCH(tv:SHOW {title: "Arrow"})
MERGE (u)-[:LIKED]->(tv)
WITH u, tv
MATCH (u)-[r:DISLIKED]->(tv)
DELETE r
```

User “yash” will dislike TV show “Arrow” which he has watched:

```
MATCH (u:USER {username: "yash"})
MATCH(tv:SHOW {title: "Arrow"})
MERGE (u)-[:DISLIKED]->(tv)
WITH u, tv
MATCH (u)-[r:LIKED]->(tv)
DELETE r
```

User “yash” wants to see the title of all the shows which he has liked:

```
MATCH (u:USER {username: "yash"})-[:LIKED]->(tv:SHOW)

RETURN tv.title
```

User “yash” liked a TV show “Arrow” but now he wants to unlike it:

```
MATCH (u:USER {username: "yash"})-[r:LIKED]->
        (tv:SHOW{title: "Arrow"})
DELETE r
```

User “yash” wants to see the title of all the shows which he has disliked:

```
MATCH (u:USER {username: "yash"})-[:DISLIKED]->(tv:SHOW)

RETURN tv.title
```

User “yash” disliked a TV show “Arrow” but now he wants to remove it from his disliked shows:

```
MATCH (u:USER {username: "yash"})-[r:DISLIKED]->
        (tv:SHOW{title: "Arrow"})
DELETE r
```

User “yash” wants to see all the shows recommended to him:

```
MATCH (r:RECOMMENDATIONS {owner: "yash"})--(tv:SHOW)

RETURN tv.title
```

User "yash" wants to watch a TV show from his recommendations:

When yash will watch TV show "The Flash" which has been recommended to him after watching "Arrow", The Flash will be removed from his recommendations.

```
MATCH (u:USER {username: "yash"})
MATCH (tv:SHOW {title: "The Flash"})
MATCH (r:RECOMMENDATIONS {owner: "yash"})
MERGE (u)-[:WATCHED]->(tv)
WITH tv, r, u
MATCH (tv)-[rel:RECOMMENDED_TO]->(r)
DELETE rel
```

User "yash" wants to remove TV show "Arrow" from the list of shows he has watched:

When yash will remove the TV show arrow from his watched shows then his like/ dislike will also be removed.

```
MATCH (u:USER {username: "yash"})
MATCH(tv:SHOW {title: "Arrow"})
MATCH (u)-[r]->(tv)
DELETE r
```

US4 Queries: Creating a watch party

User logs for creation of watch party where the details come from MongoDB:

```
db.new_login.find({new_UserName: "Praj01"}).pretty()
```

Follow a friend:

Search a friend by entering friends' username:

```
MATCH(u:USER{username:"Praj01"})
RETURN u.username
```

Follow searched friend:

```
MATCH(u:USER{username:"Praj01"})
MATCH(a:USER{username:"Naruto"})
MERGE(a)-[x:FOLLOWS]->(u)
```

Upgrade a user to premium:

Updating cash points after using them while payment:

```
db.
reward.update_one({"_id": get_id},{"$set" : {"Reward value" :
50 }})
```

Updating user to premium after payment:

```
db.new_login.update_one({"_id": get_premiumid},{"$set" : {"use
r_type" : "Premium"}})
```

Check for friend list user have:

```
MATCH (u:USER{username:"Praj01"})-[x:FOLLOWS]->(a:USER)

RETURN a.username
```

Storing

```
hset room_details Praj01_room 948bc8
```

Creating a watch party:

```
hset watchrooms_new Praj01 Praj01_room
```

Checking room details of watch party:

```
hget room_details Praj01_room
```

Adding user to that watch party after entering invitation code:

```
sadd Praj01_room Naruto
```

Checking who all are available in that watch party:

```
smembers Praj01_room
```

US5 Queries: TV shows and user management

Most liked show by users

```
MATCH (tv:SHOW)<-[r:LIKED]-
()  RETURN  "Sherlock" AS showname,  COUNT(r)AS likes order by
likes desc
```

Disliked shows by users

```
MATCH (n:SHOW)<-[r:DISLIKED]-()  RETURN  "The
Flash" AS showname,  COUNT(r)AS Dislikes order by
Dislikes desc
```

Most watched shows

```
MATCH (n:SHOW)<-[r:WATCHED]-()  RETURN  "Breaking
Bad" AS showname,  COUNT(r)AS watched order by watched desc
```

Create user:

```
mydict = {
                "new_User_Name": AnirudhAdimulam ,
                "new_First_Name": Anirudh ,
                "new_Last_Name": Adimulam ,
                "new_Email_id": anirudh@gmail.com,
                "new_User_Password": password@123,
                "new_Phone_Number": 9098564736,
                "new_Date_of_Birth": 29.04.1997,
                "new_Place_of_Birth": Hyderabad,
                "user_type" : Admin
            }
    mycol.insert_one(mydict)
```

Update user:

```
   mycol.update_one({"new_User_Name": criteria},{"$set":
{"new_First_Name":"anirudh"}})
     mycol.update_one({"new_User_Name": criteria},{"$set":
{"new_Last_Name":"adimulam"}})
     mycol.update_one({"new_User_Name":
criteria},{"$set": {"new_Email_id":'as@gmail.com'}})
     mycol.update_one({"new_User_Name": criteria},{"$set":
{"new_Phone_Number":"123444544"}})
     mycol.update_one({"new_User_Name": criteria},{"$set":
{"new_Date_of_Birth":"29.04.1997"}})
     mycol.update_one({"new_User_Name": criteria},{"$set":
{"user_type":"premium"}})
```

Delete user:

```
  mycol.delete_one("Patrick Melrose")
```

Create TV shows:

```
MERGE (tv:SHOW{title:" The Flash" , rating:toInteger("
7.2"),seasons:toInteger("6"),plot:"the flashd nkdjf  fdfjsknvo
pa",genre1:"Action ", genre2:"drama", genre3:"sci
fi", universe:"Arrow"})
MERGE (w1:WRITER{name:"OScar DSouza", dateOfBirth:"1212121",
placeOfBirth:"Hollywood",netWorth:"21Million"})
```

```
MERGE
(w2:WRITER{name:"Onkar", dateOfBirth:"0912436", placeOfBirth:"
India", netWorth:"1111Million"})
MERGE (w1)-[:WROTE{numberOfEpisodes:"69", timePeriod:"2010-
2020"}]->(tv)
MERGE (w2)-[:WROTE{numberOfEpisodes:"96", timePeriod:"2002-
2020"}]->(tv)
MERGE
(a1:ARTIST{name:"kishor", dateOfBirth:"06.06.19996", placeOfBi
rth:"telengana",netWorth:"4million"})
MERGE
(a2:ARTIST{name:"sahit", dateOfBirth:"04.04.1994", placeOfBirt
h:"hyderabad",netWorth:"3 Millon"})
MERGE
(a3:ARTIST{name:"onkar", dateOfBirth:"30.04.1994", placeOfBirt
h:"secunderabad",netWorth:"4 Million"})
MERGE (a1)-
[:ACTED_IN{numberOfEpisodes:"6", screenName:"balu"}]->(tv)
MERGE (a2)-
[:ACTED_IN{numberOfEpisodes:"6", screenName:"adimulam"}]-
>(tv)
MERGE (a3)-
[:ACTED_IN{numberOfEpisodes:"4", screenName:"anirudh"}]->(tv)
MERGE
(d1:DIRECTOR{name:"vishal", dateOfBirth:"20.05.1995", placeOfB
irth:"heidelberg", netWorth:"11 millions"})
MERGE
(d2:DIRECTOR{name:"akash", dateOfBirth:"29.05.1996", placeOfBi
rth:"mannheim", netWorth:"12 million"})
MERGE (d1)-[:DIRECTED{numberOfEpisodes:"5", timePeriod:"2015-
2020"}]->(tv)
MERGE (d2)-[:DIRECTED{numberOfEpisodes:"12", timePeriod:"2010-
1019"}]->(tv)
 MERGE
(p:PRODUCTION_HOUSE{productionHouse:"warner"})MERGE (tv)-
[:PRODUCED_BY]->(p)"
```

Delete tv show:

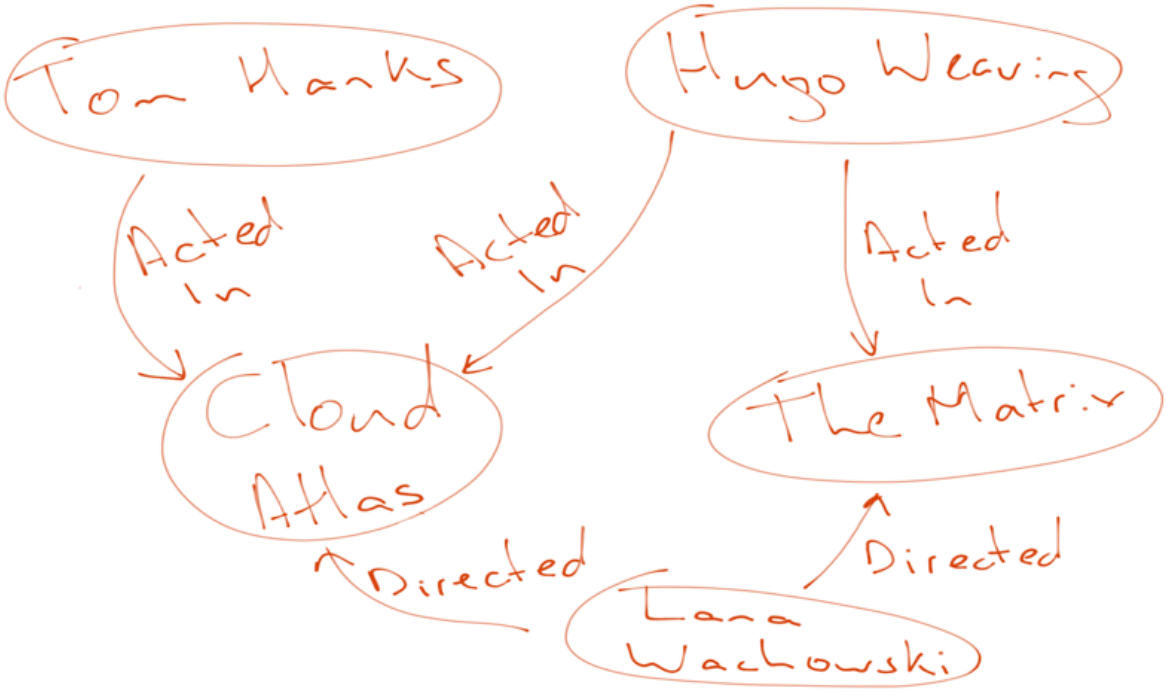
```
MATCH (n { title: "The Flash" })DETACH DELETE n
```

Implementation

In this application, we have used the graph data model, document data model and key-value data model, as well as the data of 60 tv shows including 3 artists, 2 writers, and 2 directors each, along with their dob, place of birth, net worth, genres, no. Of seasons, no of episodes, Production House, IMDB rating, Universe etc. In this section, let us study them in detail.

Graph data model:

In the Graph data model, the whole data is described visually in the form of nodes and their relationships with properties and labels. The graph data model is known to be 'Whiteboard friendly' because it is like the brainstorming and ideating sessions of teams on a whiteboard.



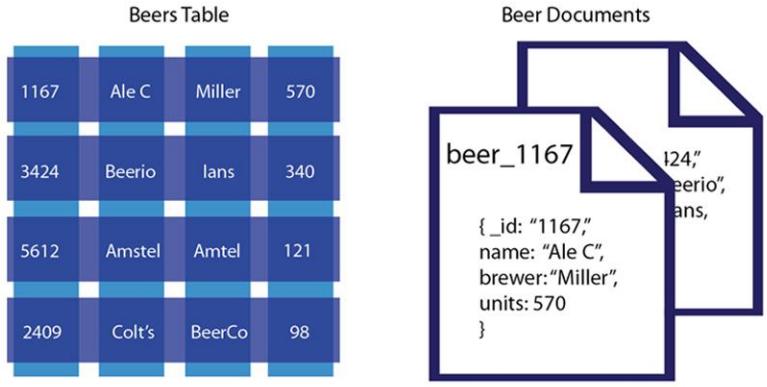
Graph data model is very flexible in terms of adaptability. It is a very good way to visualize the data and the data model. This data model is very useful in the fields of analytics, big data and data sciences, as they require detailed visualization of the data structure and model. It also comes in handy during the iterative development, where you develop data models as needed.

Document data model:

In the document data model, the data and the objects are stored in the documents. Instead of the relational database, where the data is stored in tables, in the form of rows and columns, in the document data model, it is done so in documents, in JSON-like format.

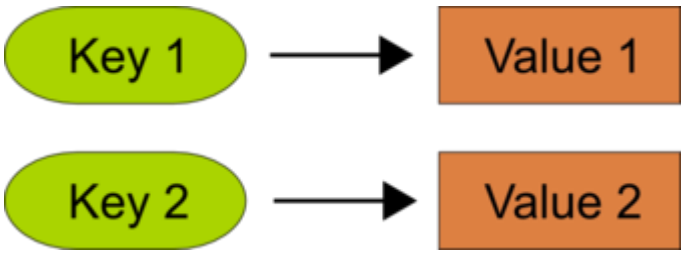
The document data model is semi structured, flexible, and hierarchical in nature, because of which the documents and document databases it makes it easier for the developers to modify their applications and databases with evolving needs of the custom. This type of data model works well with use cases such as user profiles, catalogs, and content management

systems. The reason being is that here in such use cases, each document is unique and can be easily evolved over time.



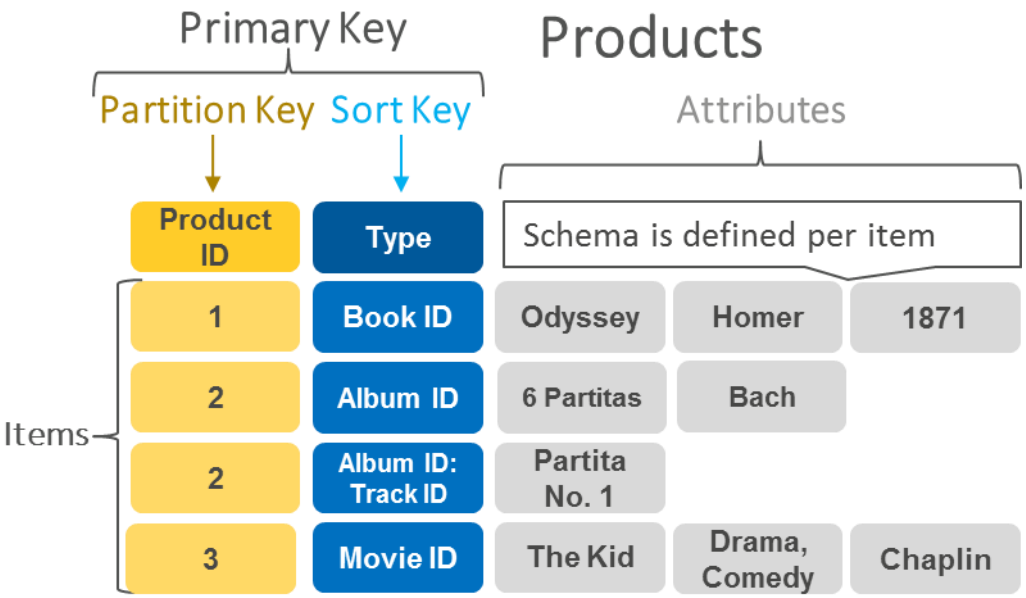
Key - Value data model:

A key – value structure of the database represents a pairing of one key with only one value. It is very similar to the dictionary data structure. Here, the key serves as a unique identifier. The key –value pairs are highly partitionable and they allow horizontal scaling which is not achievable with other database types.



Even just looking at it, we can understand that the Key-Value data model is very simple version of the database. It has a unidirectional mapping from key – to – value.

This simple data model is used in the use-cases which are more focused on the performance. This key – value data model can provide much higher performance than that of the Relational Database Management System. One more advantage of using the key – value data model is that the data modeling is very easy as compared to the RDBMS.


Primary Key
Products
Partition Key
Sort Key
Attributes
Product ID
Type
Schema is defined per item
Items
1
Book ID
Odyssey
Homer
1871
2
Album ID
6 Partitas
Bach
2
Album ID: Track ID
Partita No. 1
3
Movie ID
The Kid
Drama, Comedy
Chaplin

For this project, we have manually created a dataset – which consists of 60 tv shows and several other data fields. Following is the high-level view of our dataset.

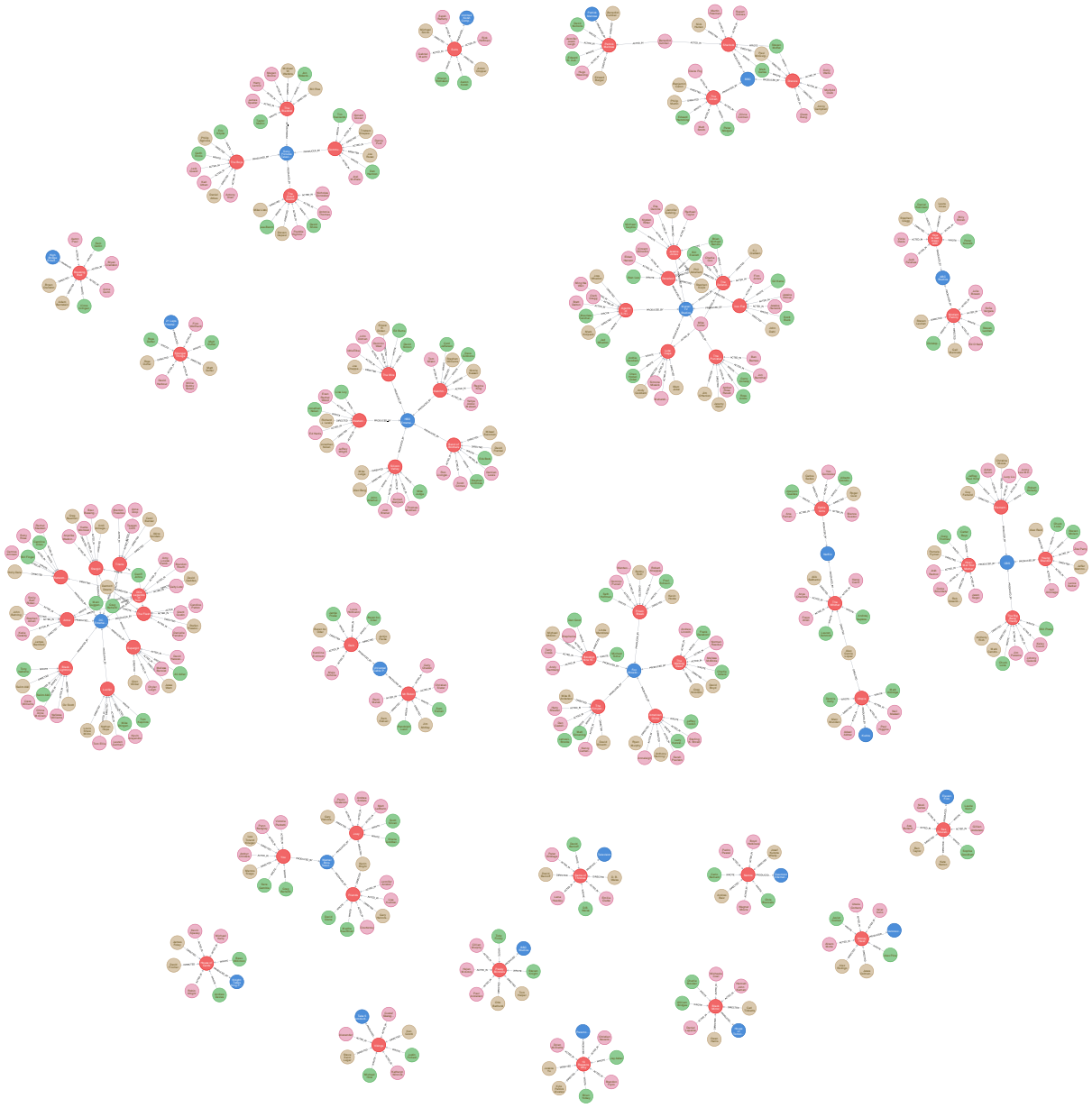


Figure 10. Neo4j Dataset

The dataset contains the following headers, which are either used to create the nodes or the relationships:

ID

Title

Artist1 Artist1DOB Artist1PlaceOfBirth Artist1NetWorth Artist1NoOfEps Artist1ScreenName

Artist2 Artist2DOB Artist2PlaceOfBirth Artist2NetWorth Artist2NoOfEps Artist2ScreenName

Artist3 Artist3DOB Artist3PlaceOfBirth Artist3NetWorth Artist3NoOfEps Artist3ScreenName

*Artist4

Director1 Dir1DOB Dir1PlaceOfBirth Dir1NetWorth Dir1NoOfEps Dir1TimePeriod Director2 Dir2DOB Dir2PlaceOfBirth Dir2NetWorth Dir2NoOfEps Dir2TimePeriod Genre1 Genre2 Genre3

Plot

Seasons

IMDBRating

Writer1 Writer1DOB Writer1PlaceOfBirth Writer1NetWorth Writer1NoOfEps Writer1TimePeriod

Writer2 Writer2DOB Writer2PlaceOfBirth Writer2NetWorth Writer2NoOfEps Writer2TimePeriod

ProductionHouse

Universe

*-Size of data was increasing so we did not use the data for Artist4

Evaluation

In the process of developing TV shows database management system, all the team members were able to work and explore three different databases determining the advantages and disadvantages of one database over other for particular use case. We decided to store User details in MongoDB as it can handle traffic more efficiently, while the entire TV shows data with all the relationships is stored in Neo4J which proved to be appropriate for our use case requirements. The decided use cases were correctly modelled and are giving desirable results with developed queries for the same.

Team members were able to efficiently use all three databases for the result of TV shows database management system. For further development, we can try to integrate our database system with a frontend.

What did we miss?

Firstly, the data synchronisation between MongoDB and Neo4j was a little complicated to implement. It was too late before we could figure out how to implement the connectivity on a standalone system and not by creating a replica set of the database.

Secondly, we could not implement this database system with a front-end application to create a full-stack application. We were planning to do this, but because of lack of time we missed out on doing it.

Bibliography

1. Redis: https://redislabs.com/get-started-with-redis/
2. Python and MongoDB: https://realpython.com/introduction-to-mongodb-and-python/
3. Python and MongoDB: https://www.w3schools.com/python/python_mongodb_getstarted.asp
4. Neo4j: https://neo4j.com/developer/cypher-query-language/
5. Neo4j Nodes and Relationships: https://pythontic.com/database/neo4j/create%20nodes%20and%20relationships
6. Neo4j Cypher query language: https://neo4j.com/developer/cypher-query-language/
7. Neo4j and Python: https://marcobonzanini.com/2015/04/06/getting-started-with-neo4j-and-python/
8. Python tutorials: https://www.javatpoint.com/python-tutorial
9. Graph data model: https://neo4j.com/blog/data-modeling-basics/
10. Document data model: https://docs.mongodb.com/manual/core/data-modeling-introduction/
11. Dictionary and Key-Value Model: https://www.pythonforbeginners.com/dictionary/how-to-use-dictionaries-in-python/
12. https://neo4j.com/developer/guide-data-modeling/
13. http://graphdatamodeling.com/
14. https://developer.couchbase.com/documentation/server/3.x/developer/dev-guide-3.0/compare-docs-vs-relational.html
15. https://aws.amazon.com/nosql/document/
16. https://medium.com/@wishmithasmendis/from-rdbms-to-key-value-store-data-modeling-techniques-a2874906bc46
17. https://aws.amazon.com/nosql/key-value/