**Section 1**

**Question 1a):** What would you implement as a 1:Many documents through linking? List two document names and what is the link between them. (short answer).

I would link the member.json and the rental.json documents. The link between the documents is member\_id. It is a one to many relationship because a member can have rented many dvds over the time period that they have been a member.

**Question 1b):** What would you implement as a 1:Few document through embedding List the document name and what would be embedded within the document. (short answer).

I would take the member.json document and embed the collection of dvd.json. I would do this because the member can only have 2 or 3 dvds out at one time, as those are the only two membership types (giving it low cardinality).

**Question 1c):** What would you implement as a 1:1 relationship through embedding within the movie rental database? (short answer)

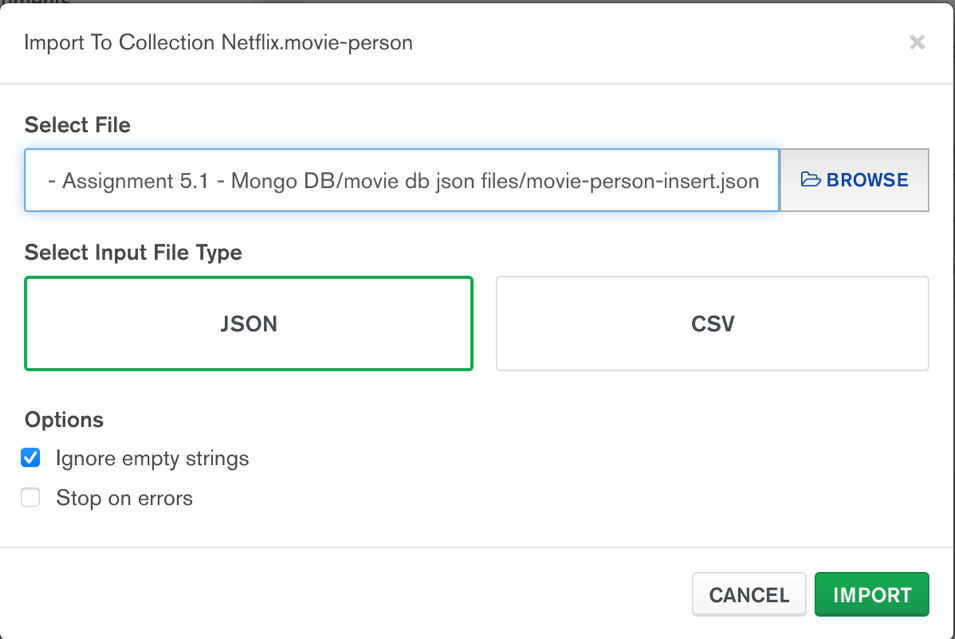
I would take the member.json and embed the movie-person.json. A member could be a movie person, and a movie person could be a member. But they can only be the same person. (e.g. Steven Spielberg could be a Netflix member, he as a Netflix member who is also a movie person can only be one movie person).

**Section 3**

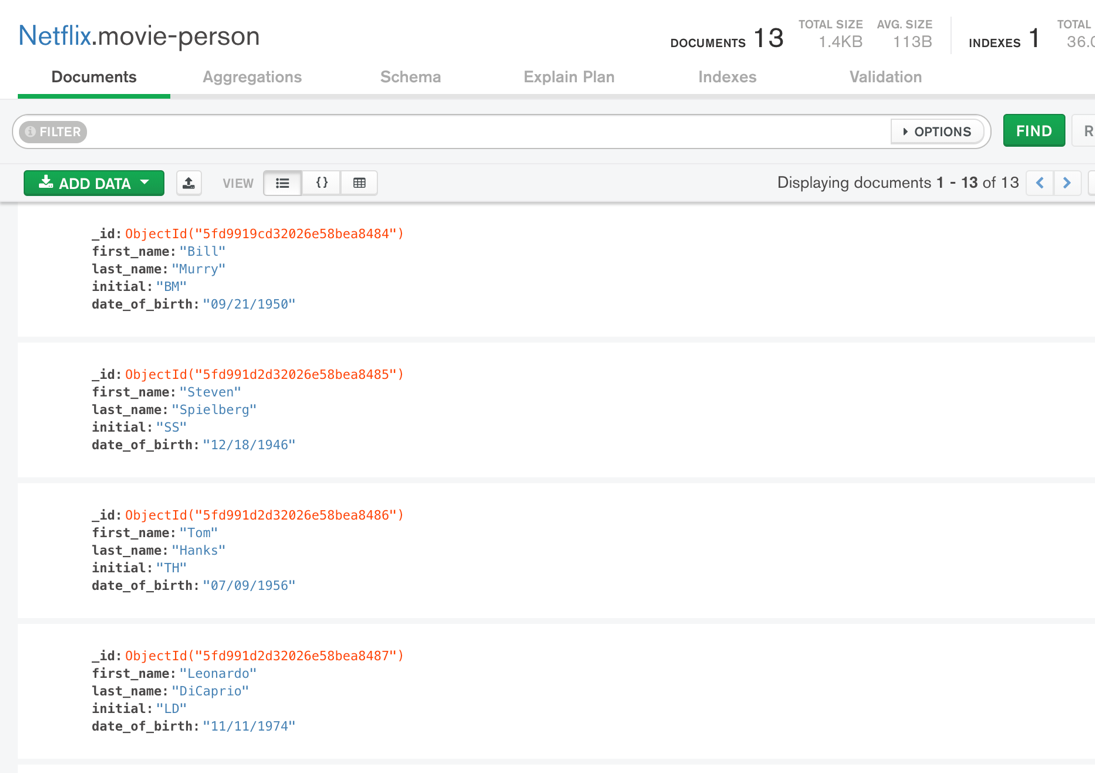
**Question 3a):** Import the rest of the movie people by using either the copy paste method or importing the JSON document.

Paste screenshot showing both copy and paste or importing the JSON document and the inserted data result. The screenshot of the result should show how many total documents are in the movie-person collection.

Importing movie-person-insert.json



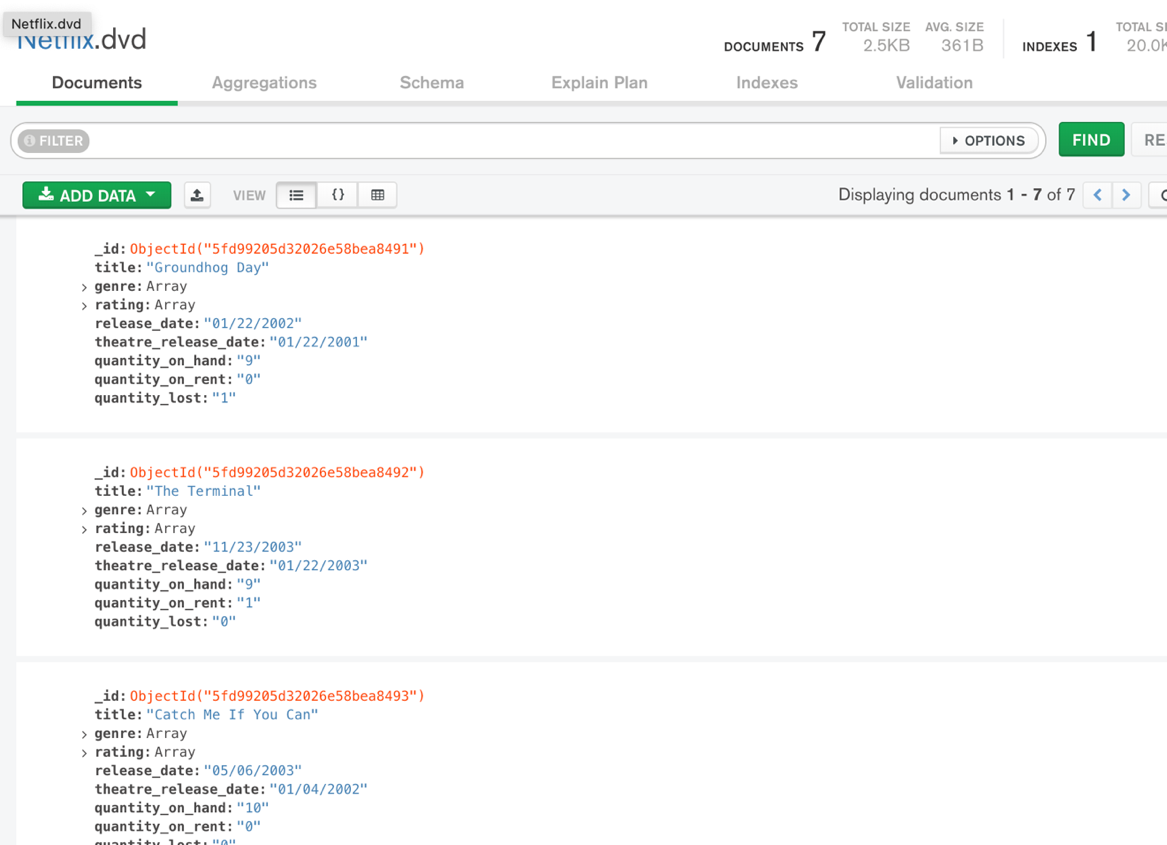
Results showing a total of 13 documents in the movie-person collection.

****

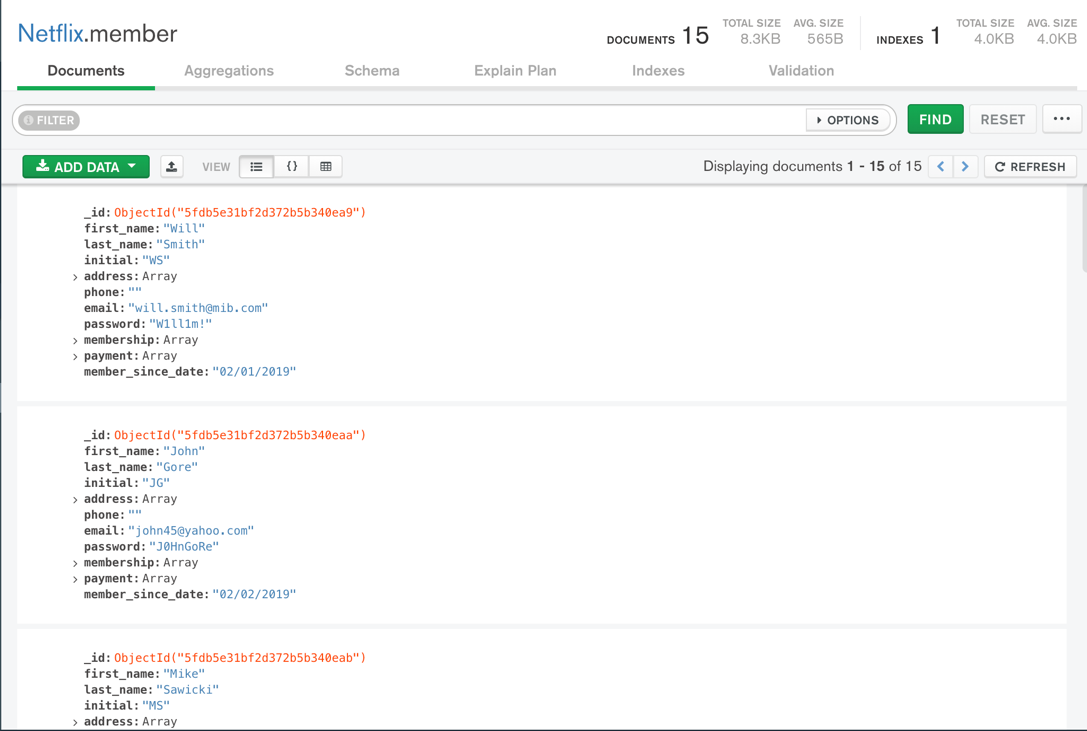
**Section 6**

**Question 6a):** Provide screen-shots of both dvd and member collections loaded with the documents from the provided JSON files, make sure to show the record counts.

dvd-insert.json import showing 7 documents in the dvd collection

****

member-insert.json import showing 15 documents in the member collection

****

**Question 6b):** Examine the DVD document and specify how the Genre and Rating relationship was implemented, was it through embed or linking? What is the cardinality of these relationships and does it correctly support the design choice? (short answer)

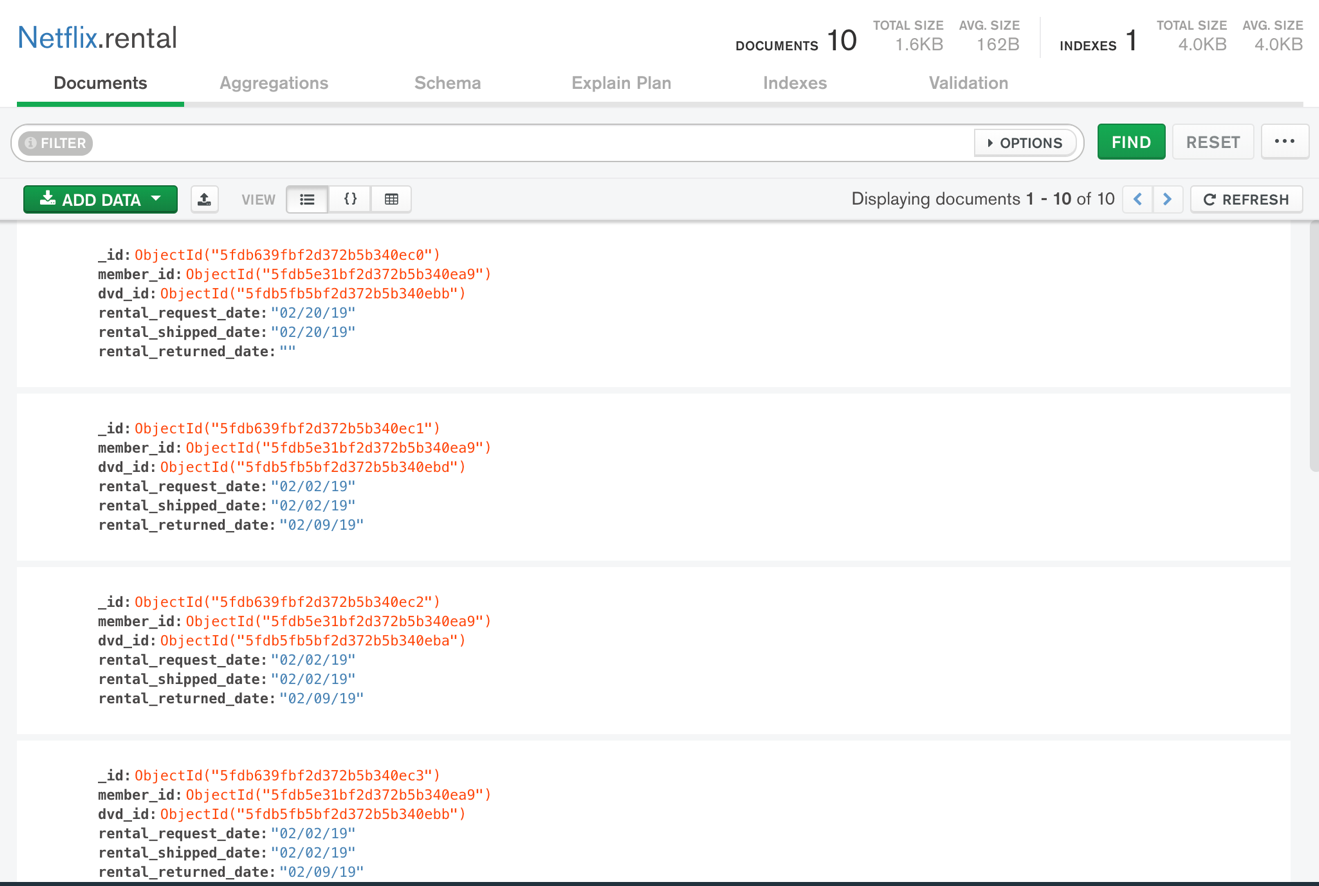
The genre and rating relationships were implemented by embedding those collections in dvd. The cardinality is one to few, as a movie could have multiple genres (e.g. a romantic comedy). It does correctly support the design choice given that movies are not typically listed with more than 2-4 genres (a few).

**Question 6c):** How are Member, DVD and rental document related? (through embdding or linking, explain your answer briefly)

Member and rental are linked, as one member can have many rentals, showing a 1 to many relationship. They are linked by id in member and by member\_id in rental. DVD is embedded in rental as a rental can only have one dvd\_id in it and only one copy of a dvd is being sent in a rental, showing they have a 1 to 1 relationship. Member and DVD are not linked and are not embedded with each other in any way.

**Question 6d):** Provide a screenshot of the populated rental collection

rental-insert.json import showing 10 documents in the rental collection



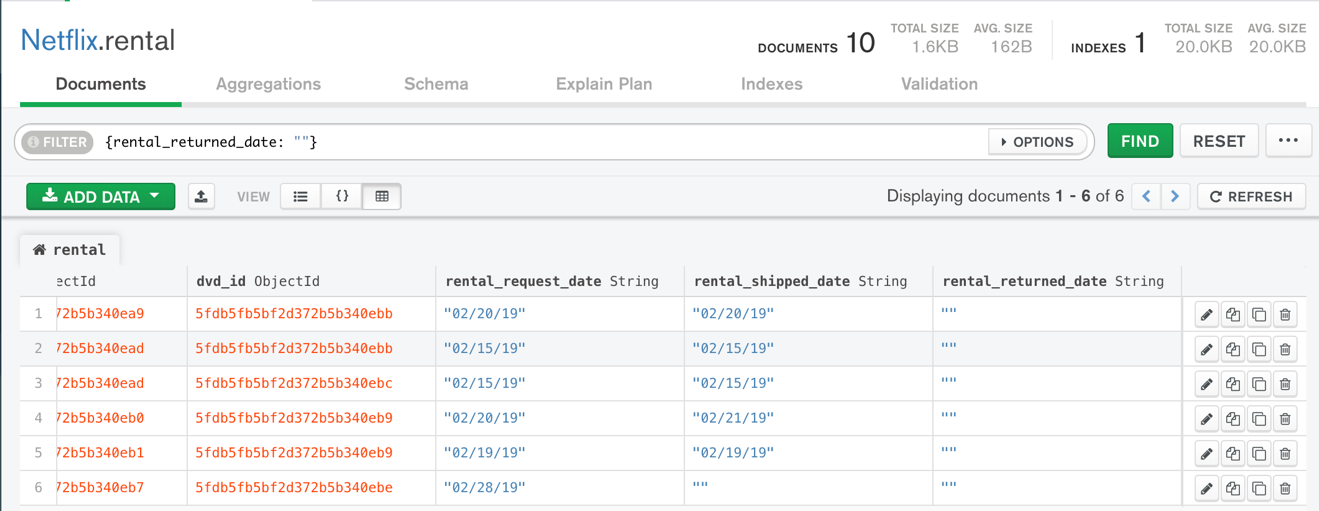
**Question 6e):** Examine the dvd collection and movie-person collection. We have not established a relationship between them. Briefly describe how would you establish the relationship i.e. via linking or embedding and what kind of a relationship is it.

I would establish the dvd collection and movie-person collection by linking them as a one to many relationship, as one dvd can have many movie persons in it.

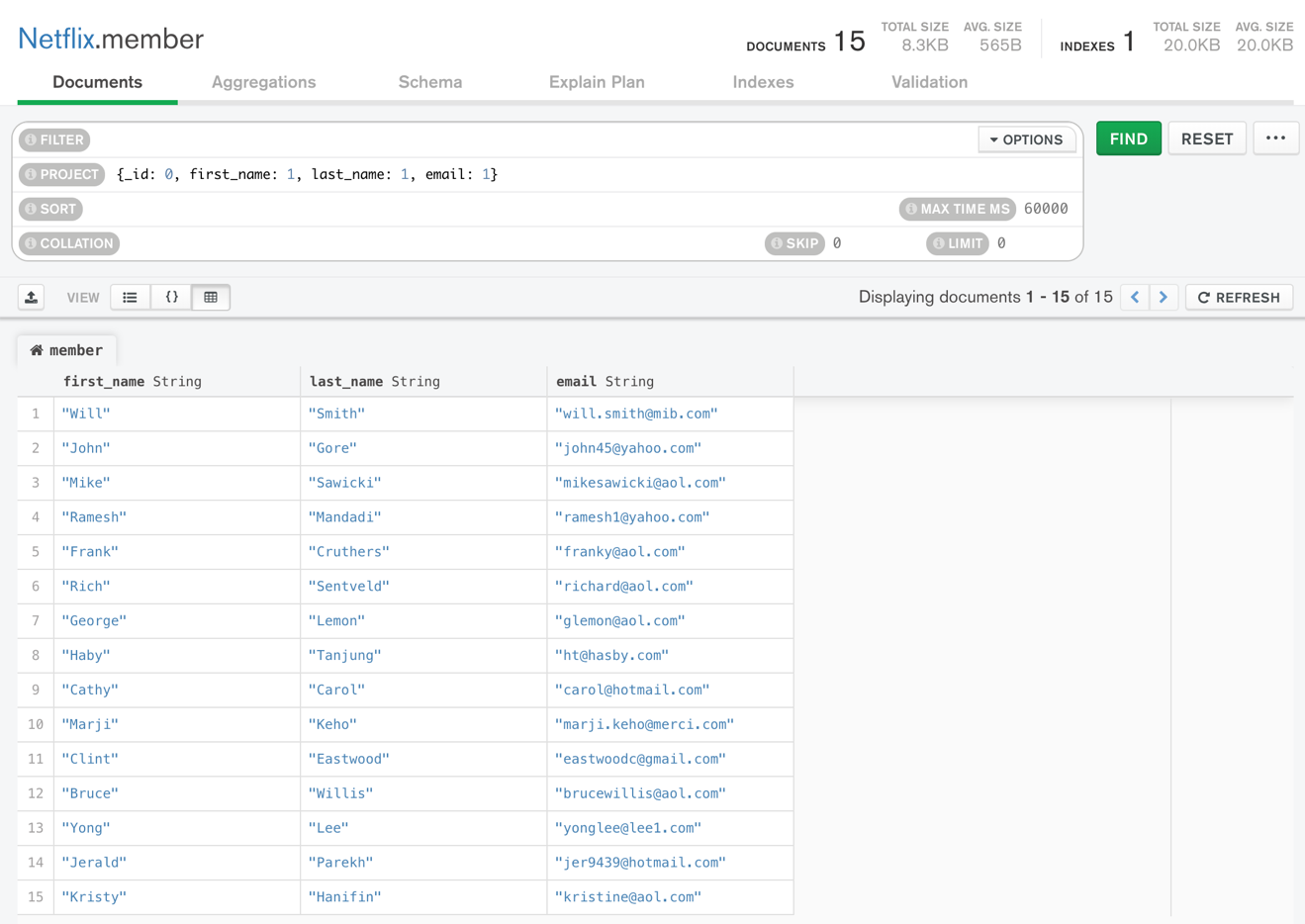
**Question 6f - extra credit):** Create a linked collection for DVD and Movie-Person called movie-person-dvd and populate it.

**Section 7**

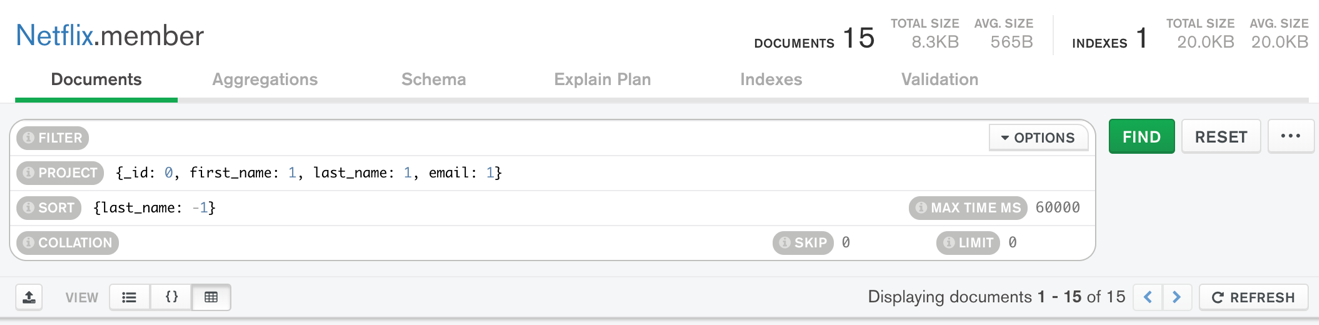
**Question 7a):** Find all the rental documents for DVDs that have not been returned. Take a screenshot of the result.

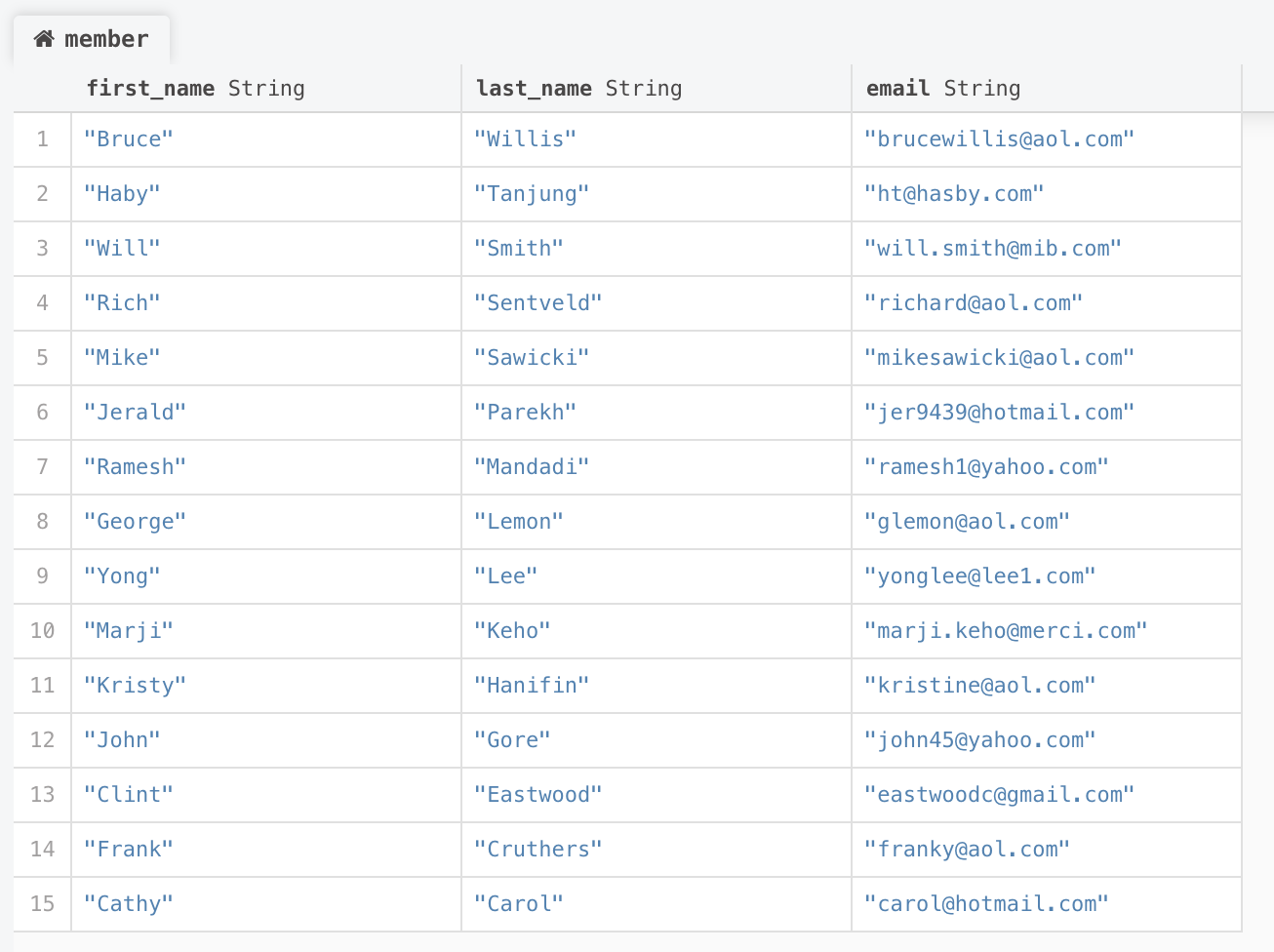


**Question 7b):** Find the first name, last name and email of all the members. Please make sure that the result does not have the \_id field. Take a screenshot of project command the result.

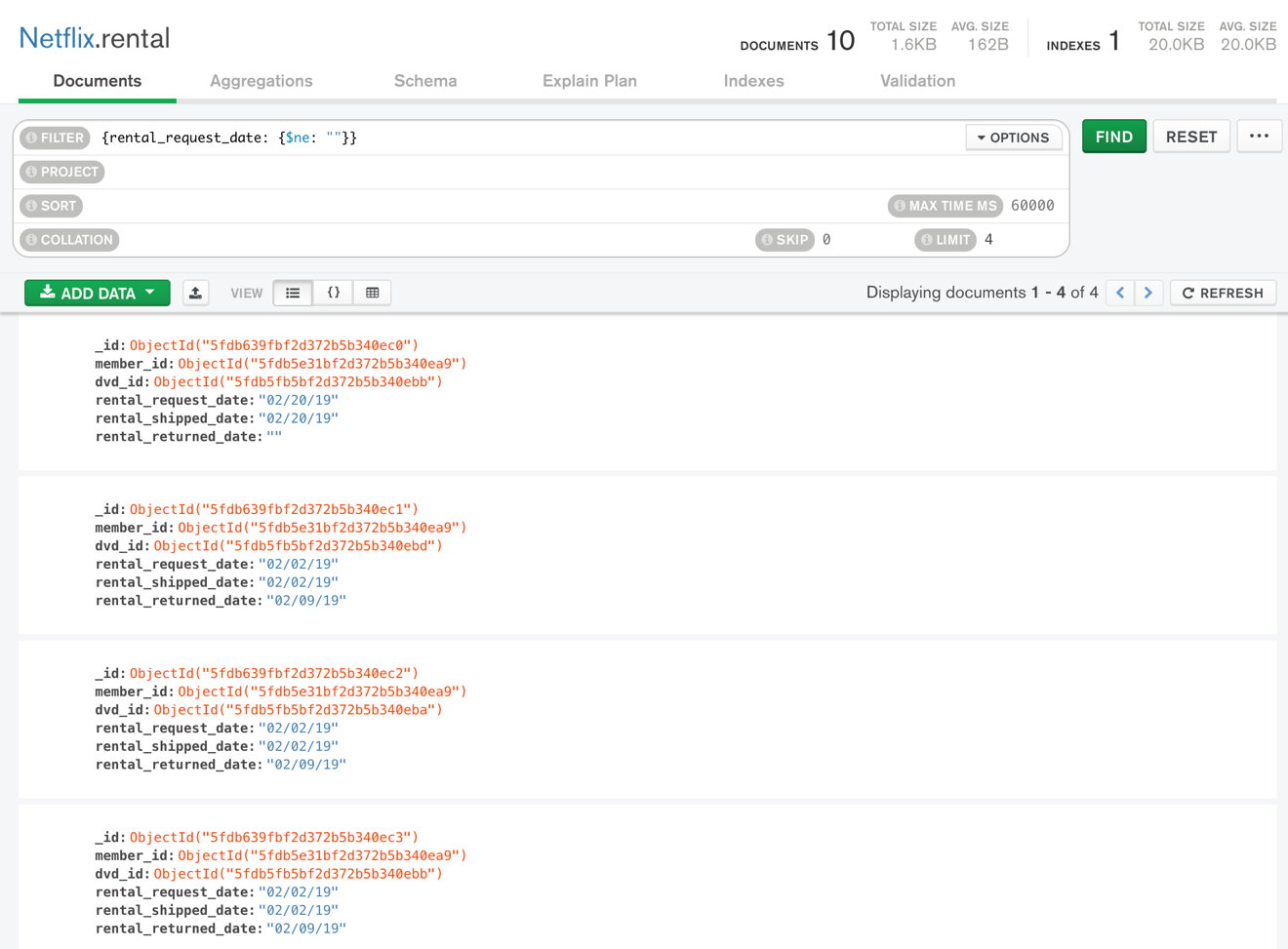


**Question 7c):** Sort the documents in the member collection based on the last name of the members in descending order. Take a screenshot of the sort command and the result.





**Question 7d)** Find 4 documents of the rental collection where the DVD is not returned by the member. Take a screenshot of the result and attach it to the final submission.



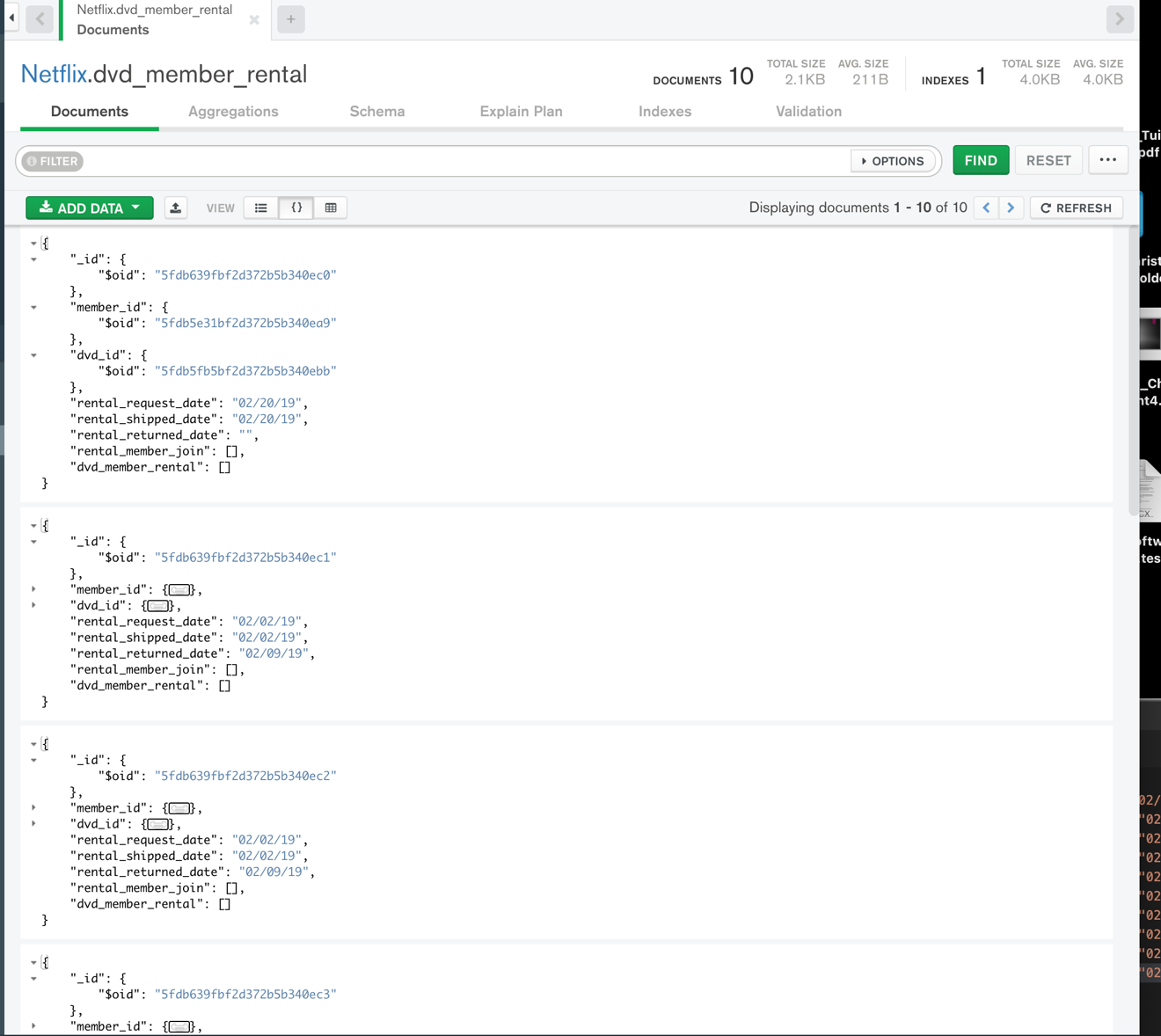
**Section 8**

**Question 8a):** Perform a left outer join between the rental and member collection and name the resulting collection as **rental\_member\_join**. Take a screenshot of the JSON structure of the last document (i.e. 11th document) from the rental\_member\_join collection. Please expand all the fields so that the values are visible clearly.



**Question 8b):** We want to access the dvd and member data together for each rental document. Which collections would you join to achieve this? Perform the join and take a screenshot of the resulting collection.

I would join the dvd collection with the rental\_member\_join to make dvd\_member\_rental.



**Question 8c):** Examine the resulting collection from Question 9. Briefly describe the problem with this collection.

It is only giving the object IDs of the joined collections, not the other information. it’s doing this because it’s only joining based on these fields.