NSQ1 S25 Course Assignment 2

# Question 1 – Revise model

If necessary, revise the model from course assignment 1. Otherwise, use the same model.

# Question 2 – Model database

Design a MongoDB model for the bookstore model from the 1st course assignment. Use design patterns where appropriate and beware the anti-patterns. Make note of the choices you make and why. Document the model in one of the 4 ways shown in the slides.

Design the model to be used with the queries from the 1st course assignment.  
  
**In create\_schemas.mongodb  
In populate\_collections.mongodb  
In queries\_assignment1.mongodb**

# Question 3 – Create schemas and indexes

Design and create schemas for the collections in your model. Implement indexes where it seems appropriate.  
  
**In create\_schemas.mongodb**

# Question 4 – Work with data

Answer the following questions in MongoDB using your model from question 1.

## 4a. Modifying data

Use MongoDB CRUD methods to execute the following scenarios. Use transactions where necessary.

If nothing else is stated, assume you know the object ids of the objects involved.

1. Sell a book to a customer.
2. Change the address of a customer.
3. Add an author to a book.
4. Retire the "Space Opera" category and assign all books from that category to the parent category. Don't assume you know the id of the parent category.
5. Sell 3 copies of one book and 2 of another in a single order  
     
   all the code is in **questions4.js**

## 4b. Querying data

Write find() or aggregate() statements to return the following data

1. All books by an author
2. Total price of an order
3. Total sales (in £) to a customer
4. Books that are categorized as neither science fiction nor fantasy (**Note:** This is harder than you think. A book can have more categories. Make sure you don’t return books that are fantasy romance, for instance.)
5. Average page count by genre
6. Categories that have no sub-categories
7. ISBN numbers of books with more than one author
8. ISBN numbers of books that sold at least X copies (you decide the value for X)
9. Number of copies of each book sold – unsold books should show as 0 sold copies.
10. Best-selling books: The top 10 selling books ordered in descending order by number of sales.
11. Best-selling genres: The top 3 selling genres ordered in descending order by number of sales.

**Note:** In the next three exercises, subcategories of science fiction also count as science fiction, and so do subcategories of subcategories, and so on. It’s not enough to just look for the science fiction category.

1. All science fiction books. Note: Books in science fiction subcategories like cyberpunk also count as science fiction. Don’t use your knowledge of the concrete category structure. (Depending on the model you might want to use $graphLookup)
2. Characters used in science fiction books.
3. For each category: Number of books in the category including books in its subcategories.  
     
   all the code is in **questions4.js**

# Question 5 – Cloud

1. Create a cluster and a database in MongoDB Atlas. Create a collection for each of the collections in Question 2.
2. Create a new collection, orderLog, and a trigger on creation of orders. The trigger should add objects to the orderLog whenever an order is inserted or updated.
   1. When inserted: The object should have a single property, “inserted”, with the order object as a value.
   2. When updated: The object should have an “updated” property with the new order, and an “updateDescription” property with the [update description](https://www.mongodb.com/docs/manual/reference/change-events/update/).
3. Implement the indexes from question 3.
   1. Already implemented

Document this with

1. The code of the trigger function

The trigger has been implemented through the MongoDB Atlas UI (enabling Full Document option)  
  
exports = async function (changeEvent) {

try {

const orderLogCollection = context.services

.get("NSQ")

.db("assignment2")

.collection("orderLog");

if (changeEvent.operationType === "insert") {

await orderLogCollection.insertOne({

inserted: changeEvent.fullDocument,

});

} else if (changeEvent.operationType === "update") {

await orderLogCollection.insertOne({

updated: changeEvent.fullDocument,

updateDescription: changeEvent.updateDescription,

});

}

return { success: true };

} catch (error) {

console.error("Error in trigger function:", error);

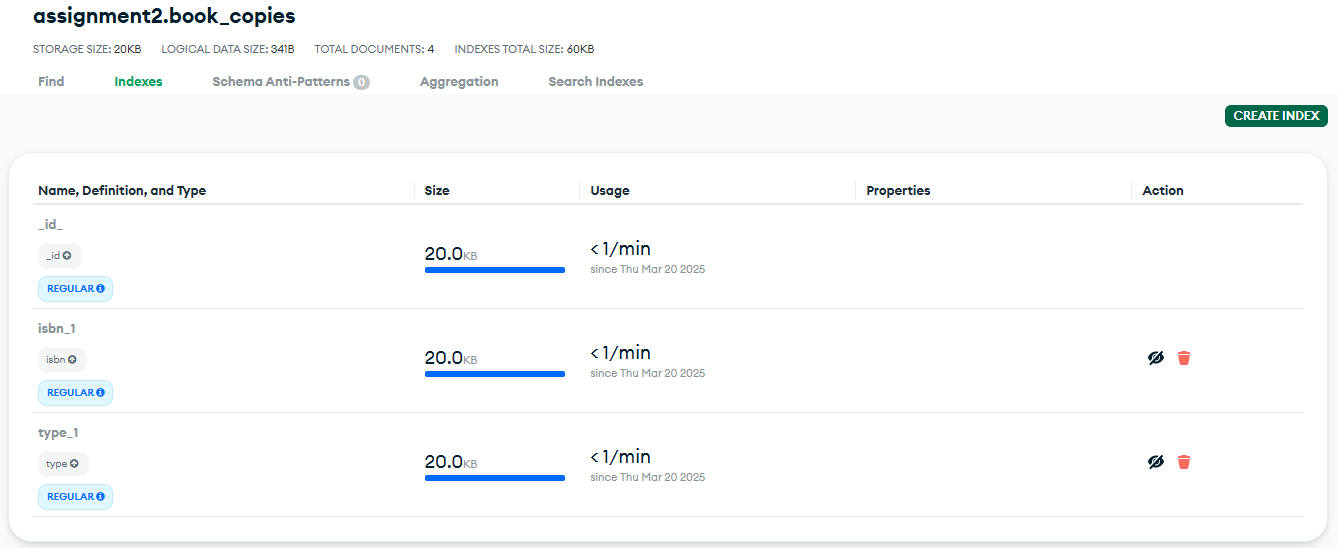
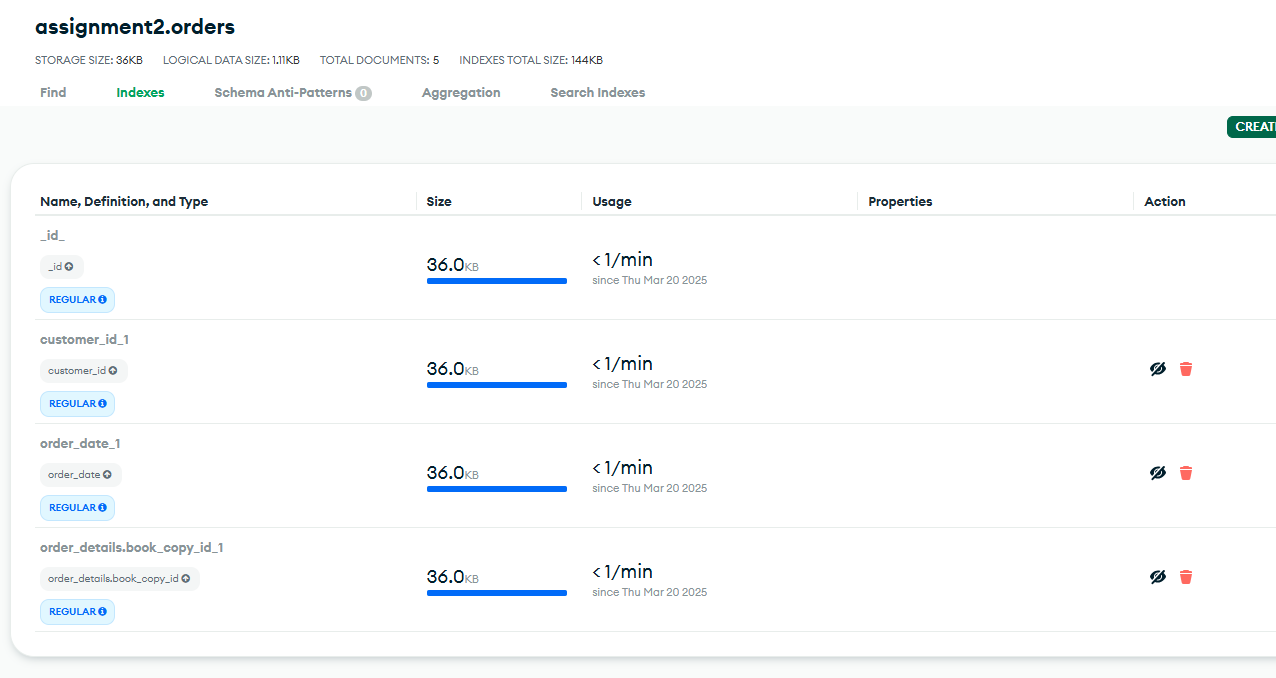
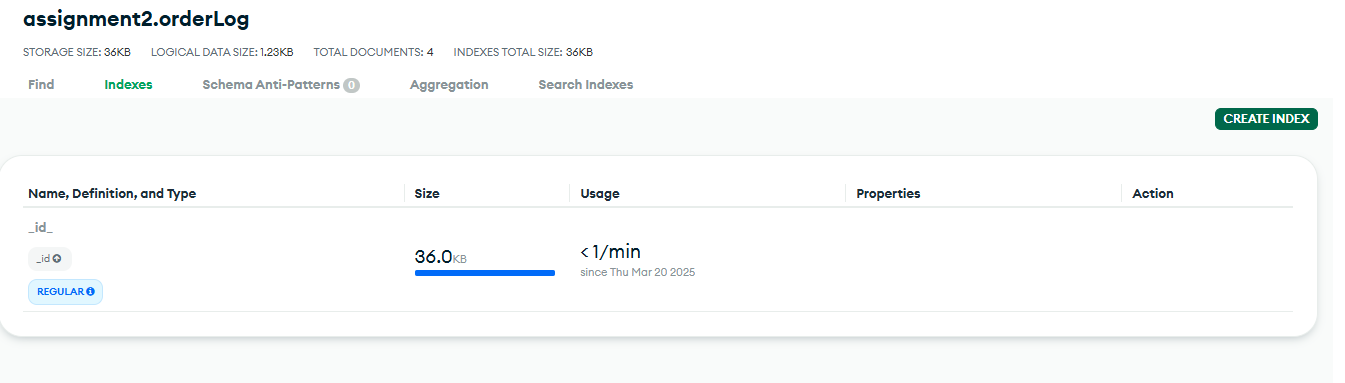
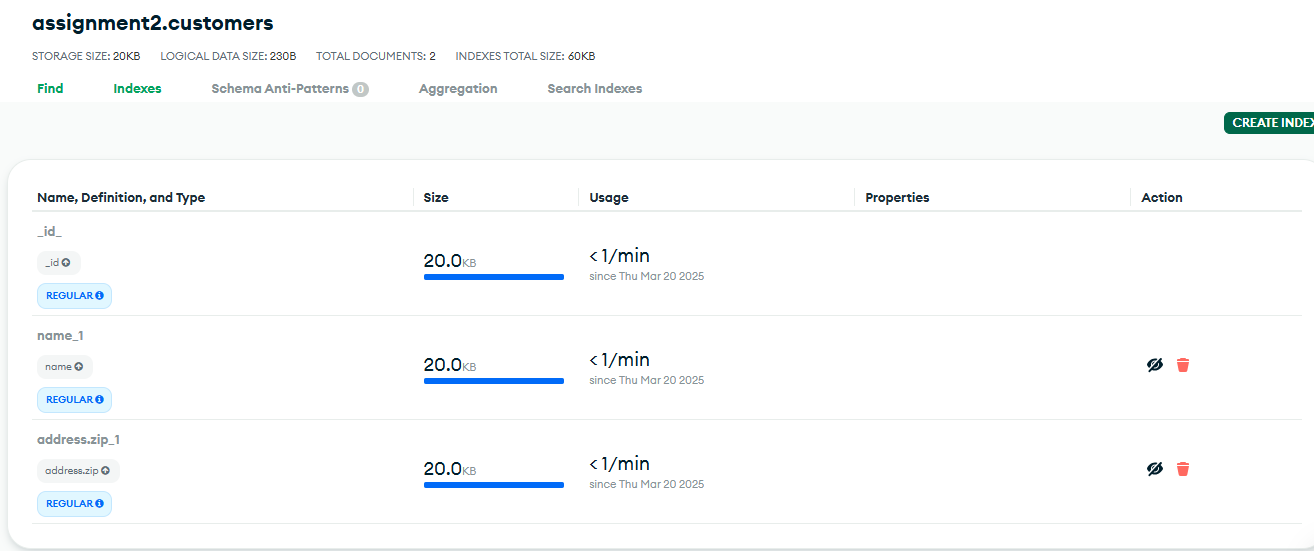
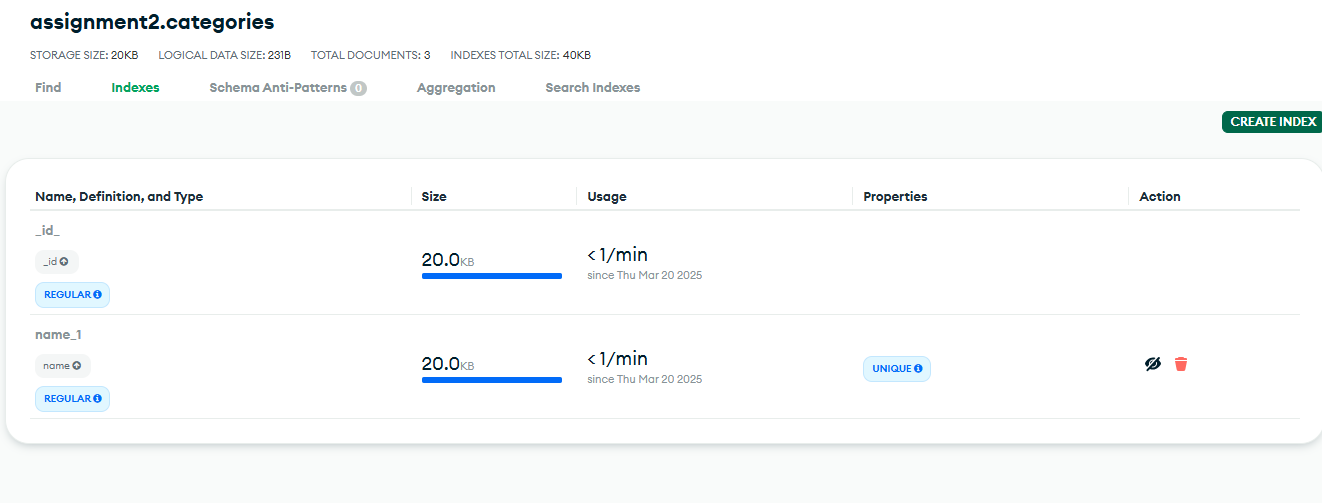
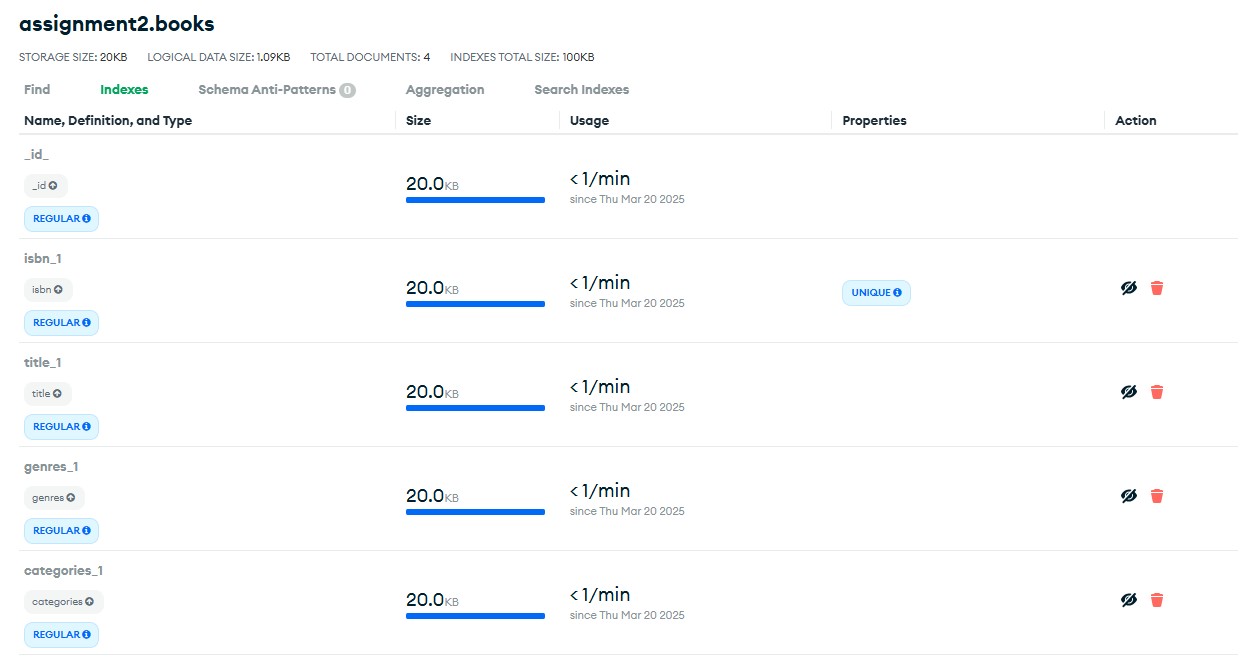
return { success: false, error: error.message };

}

};

1. Screenshots of the implemented indexes

All the indexes are shown in **create\_schemas.mongodb** file.  
once installed the extension on vs code, we connected with the cluster, executed the files and everything got stored on the cloud.A screenshot of a computer

AI-generated content may be incorrect.

# Question 6 – Report

Write a report on the experience gained by completing Question 1 and 2 above. The report should contain answers to the questions

* What were the decisions taken in the modelling?
  1. Referencing vs Embedding:
     + Authors referenced in books using ObjectId to avoid data duplication
     + Customers referenced orders using customer\_id to have a logical separation between customers and their purchase history
     + order\_details was embedded in orders to improve efficiency
  2. Data Validation and Schema Enforcement:
     + We implemented validation rules using $jsonSchema for data integrity.
     + bsonType has been used to make sure we use the appropriate data types.
  3. Indexing for performance Optimization:
     + We created indexes on frequent queried fields to speed up lookups
  4. Polymorphic Pattern for book copies:
     + Book\_copies has a field called “type” to allow different book formats (Hardcover, Paperback, E-book)
* Why were these decisions taken?
  + - Embedding was used to access data quickly. It was important for order\_details so we could avoid multiple queries when retrieving order history.
    - Referencing was useful in situations where data consistency and normalization were important, like linking books to authors.
    - Indexing was used to be efficient in query execution.
    - The validations acted as guard to prevent inconsistent or malformed data from being stored
* What were the consequences of these decisions?
  + - Improved read performance: embedding order\_details in orders allowed fetching order information in a single query.
    - Increased storage space: because of embedding there were a few redundancies, but it was worth it due to faster reads.
    - Better data integrity: using references for authors ensured consistency across different books.
    - Simplified queries: the structure of the schemas made queries more intuitive, reducing the need for complex joins.
* What were the difficult and easy parts of the exercise?
  1. Easy
     + Creating collections and defining validation.
     + Implementing indexing.
     + Implementing the trigger.
  2. Difficult
     + Deciding between embedding and referencing.
     + Ensuring update trigger stored the correct structure
* How does that compare to relational databases?
  + - Schema flexibility: mongoBD is more flexible in schema design, no need of strict table structures.
    - Joins vs embedding: in relational databases we often need joins to retrieve data, while in MongoDB embedding simplifies this problem.
    - Transactions: MongoDB supports ACID transaction, but relational databases provide more robust transactional integrity.
    - Normalization: relational databases emphasize normalisation, while MongoDB denormalizes for better performance.
* What are the advantages and disadvantages of MongoDB over relational databases for this exercise?
  1. Advantages
     + Faster reads: embedded documents allow us to fetch related data in a single query.
     + Scalability: mongoDB’s horizontal scaling is very useful for handling large datasets.
     + Flexible schemas: you can modify something without altering the entire database structure.
  2. Disadvantages
     + Redundancy: embedding data can lead to increased storage requirements.
     + Complex updates: updating deep nested fields requires careful query structure.
     + Lack of strong relations: creating strict relationships can be challenging

# Rules

* Make the exercise in groups of 2 – 4
* Hand in to itslearning no later than 23 March