MongoDB Schema Design Patterns

Inheritance Pattern

- The **Inheritance Pattern** helps us group similar documents within a collection.
- It is based on **Polymorphism** which means that each entity within the collection has a different form.
- Different forms of the same entity can exist within the collection.
- A parent entity can exist which has fields shared across all its child entities.
- Simultaneously, those very child entities can also contain unique fields.
- The Inheritance Pattern is used when there are documents with more similarities than differences and we want to keep them in the same collection so that they can be read together.
- Here, we have an example of several entities sharing common fields such as **Ebooks, Audiobooks** and **Printedbooks**. All of these entities are stored in the **Books** collection.
- Each of these entities have a field called **product_type** in their respective documents that tells the app what fields to expect & the shape of the document.
- There is a Data Transformation Pipeline called the Aggregation Framework in MongoDB. It can be used to transform a large number of documents or an entire collection of documents into the **inheritance pattern**.
- An example of using the Aggregation Framework is to use it to add **product_id** and **product_type** fields to all documents in the **Books** collection that do not specify what type of book they are: (Ebook, Audiobook or Printedbook).

Computed Pattern

- Running compute operations repeatedly can be computationally expensive and slow down the application.
- A solution to this is the **Computed Pattern** which runs the required compute operations when the data changes. It allows us to store the result for quick access.
- There are many types of computations, but we will focus on Mathematical and Rollup operations.
- For mathematical operations, the goal is to **pre-compute** the operation before the data is written instead of running the operation every time we need it. In the books application, a **Review** document consists of an SKU and a **Rating** field among others. When a customer drops a new review with a rating, a new Review document is added to the Reviews collection. Then, the average rating is calculated by multiplying the **existing average rating** with the **existing review count**. The result of this multiplication is added to the **new rating**. The result of this addition is then divided by the **current total number of reviews**. This is the **new average rating**. It is stored in the corresponding book document in the **rating** sub-document.
- **Another type of computed pattern is Rollup Operations**.
- It allows us to view data as a whole.
- One use case is grouping categories from smaller units into larger ones such as hourly, daily, monthly or yearly reports.
- Eg - We need the total number of books by product type and the average number of authors in a given product type.

- Instead of computing this info every time we need it, we can store this info in a document and update it at some intervals.
- This roll-up operation will be implemented using an aggregation pipeline.
- We need to roll up a summary document for each product type which includes the total number of books and average number of authors in that product type.

Approximation Pattern

- This pattern allows us to get a number which isn't exact but very close to the exact number.
- We use this pattern when data is either difficult to get or calcs are expensive and the exact number isn't critical to our use case.
- Well suited for working with big data.
- It reduces and in some cases it can reduce contention on heavily written documents.
- **This pattern is implemented in the application logic**.
- An example of this would be in counting the number of reviews that a book has received.
- If the book has received a very large number of reviews (100000 or million), then we should calculate the number of reviews only periodically and not on demand (every time the page loads).
- This would significantly reduce the number of write operations.
- One way to implement this in the books app is to have an RNG that can generate a random number between 1 & 10, when a new review is posted. If the generated number is 10, then we update the number of reviews and the average rating.
- We update the total reviews by adding 10. When computing the new average rating we use the new rating value multiplied by 10. This will give us a number of reviews and an average rating that aren't exact but statistically correct.
- We will need to store the approximate number of reviews and approximate average rating in a separate sub-document in the corresponding book document.
- This logic will be implemented in the application logic and will not affect our data model.

Extended Reference Pattern

- This pattern helps us to get information from documents in various collections without having to use the aggregation **$lookup** operator.
- An extended reference is a reference that is rich enough to include all that is needed to avoid a join.
- An extended reference pattern can be created by embedding relevant data from multiple documents and collections into the main document.
- Its primary objectives are to reduce the latency of read operations, avoid round trips to the database and avoid touching too many pieces of data.
- It leads to faster reads due to fewer joins.
- Data duplication should be minimised while using this pattern.
- This pattern works best if the fields selected to be embedded in the main document, are those that don't change often.
- Duplicated data needs to be updated along with the source.
- When a source field is updated, we need to create a list of dependent external references (duplicated fields) that need to be updated. Such updating may not need to be done immediately.

Schema Versioning Pattern

- This pattern is used to change the existing schemas of the database.
- By adding an extra field to our schemas and using some application support, it is possible to change schemas with no downtime.
- In MongoDB, schemas with different shapes can exist in the same collection.
- The schema version number field in a document helps MongoDB identify the shape of the schema.
- Every document in a MongoDB collection can have a unique schema number that helps uniquely identify its shape. It is stored in the **schema_version** field.
- If the schema version number isn't assigned, then its default value is 1.
- In practice, during a schema transition, only few versions of a particular type of schema will exist in a collection.
- Before updating the schema, the application must be updated so that it can read all schema versions: old & new.
- There are 2 ways to update the schema in a collection: Have the application update the document schema when the document is accessed or have a background process update all the schemas.
- **The schema update sequence is given below**:
- We begin by updating the application servers.
- The documents are updated by running the schema update task.
- While the documents are updating, the app can handle both versions of schemas.
- The background schema update task can also be run with multiple passes. It is best to run it when the system is less busy.
- Once the update is complete, the code to handle old versions of the schemas can be removed.

Non-Video

Building with Patterns: The Single Collection Pattern | MongoDB
Building with Patterns: The Subset Pattern | MongoDB
Building with Patterns: The Bucket Pattern | MongoDB
Building with Patterns: The Outlier Pattern | MongoDB